Universidad
Carlos III de Madrid

Estación de tierra autónoma para la gestión de telemetría en Vehículos Aéreos no Tripulados (UAVs)

PROYECTO FIN DE CARRERA

Ingeniería Industrial, especialidad en Automática y Electrónica

Realizado en Universitat Politècnica de València

Departamento de Ingeniería de Sistemas y Automática

Autor:
Juan Enrique López Carcelén

Director:
Javier Sanchis Sáez
Sergio García-Nieto Rodríguez

Co-director:
Basam Musleh Lancis

Valencia, Junio de 2013

Este libro contiene los siguientes documentos:

AGRADECIMIENTOS

En primer lugar, agradecer a mi director de proyecto Javier la posibilidad de trabajar en este proyecto que me ha aportado mucho tanto en lo académico como en lo personal. Durante la realización de mi trabajo, he de agradecer a mi supervisor Sergio toda su paciencia y apoyo continuo desde el primer día. Además, agradecer a Jesús por todo su tiempo y desearle mucha suerte en su futuro como doctorando.

Quiero agradecer también el apoyo recibido por la gente que he conocido en mi Séneca y que ha estado muy cerca de mí este año, especialmente a Celia, Pedro, Laura, Pepe, María, Sandra y Joaquín, sin los cuales este último año de mis estudios de ingeniería no se hubiese convertido en uno de los mejores.

Por último y lo más importante, agradecer el calor durante todos estos años de mi familia. Especialmente a mi tía María Elena, a mi tío Quique al cual considero un padre, a la persona que me ha ayudado a alcanzar lo que soy hoy, mi madre. Y sobre todo dedicarle este pequeño logró en mi carrera a mi abuela Carmen, que siempre estará con nosotros.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

En el entorno de los aviones aéreos no tripulados (UAV), una de las partes más importantes es la telemetría del avión. Gracias a ella se consigue localizar, monitorizar y extraer medidas en tiempo real relevantes para el estudio de la aerodinámica del aparato. Para que la transferencia de datos sea adecuada, hace falta tener en cuenta muchos factores como por ejemplo la calidad del envío de la señal, la banda de frecuencia en la que se trabaja, las interferencias con otros dispositivos, etc. Dependiendo del tipo de antena en la estación de control de tierra utilizada para la comunicación con el avión, puede ser importante que la orientación de dicha antena apunte al UAV con el fin de conseguir la mayor potencia de transmisión posible.

1.1.1. Infraestructura previa al proyecto

Basada en esa idea, y partiendo de una plataforma de vuelo existente y una estación de control de tierra en desarrollo esquematizada en la Figura 1.1, surge la idea de implementar un sistema de gestión para que la antena sea capaz de llevar a cabo esta tarea de forma autónoma.

Figura 1. Esquema del sistema inicial

En la figura anterior se trata de esquematizar el sistema inicial en el que se pretenden implementar las mejoras que constituyen el núcleo de este proyecto. Previamente la antena es controlada manualmente por un operario para que esté continuamente apuntado al UAV. Y por otro lado también representa la conexión física que existe con el sistema de monitorización que en este caso se comunica por conexiones puerto serie (RS-232).

1.2. Objetivos del proyecto

El objetivo fundamental del proyecto es implementar una estación de control de tierra autónoma para la gestión de la telemetría de un vehículo aéreo no tripulado. Para llevar a cabo esta finalidad se han propuesto los siguientes objetivos por orden de prioridad:

- Orientar permanentemente la antena hacia el UAV
- Independizar la antena del sistema de monitorización
- Implementar una interfaz gráfica con el usuario mediante un servidor web
- Implementar un sistema de rastreo en caso de pérdida de localización del UAV
- Posibilitar la monitorización de la telemetría desde más de un ordenador
- Visualizar de la orientación de la antena en tiempo real en el mapa de vuelo

1.2.1. Orientar de permanentemente la antena hacia el UAV

En primer lugar y lo más importante es automatizar el movimiento de reorientación de la antena. La antena posee dos grados de libertad rotativos y dos servomotores que gobiernan estos dos movimientos de rotación. Obteniendo la posición global del UAV por transmisión de radio se debe programar una aplicación capaz de controlar los servomotores de manera que la antena se oriente hacia la posición en la que está el avión.

1.2.2. Independizar la antena del sistema de monitorización

El segundo objetivo es hacer que la antena sea independiente de una conexión física con el ordenador que visualiza los parámetros recibidos por la telemetría del avión y

que a su vez también puede servir para enviar señales de control al UAV en el otro sentido de la comunicación. Este dispositivo está en el otro extremo del sistema global y es donde se interacción con el usuario. De ahora en adelante, nos referiremos a este subsistema como punto de monitorización.

Figura 2. Antena sin operario y desvinculada del PC de monitorización

Del esquema de partida, se prescinde tanto del operario que controla manualmente la antena gracias al control autónomo, como de la conexión física mediante puerto serie (RS-232) entre el punto de monitorización y la antena gracias a la tecnología wireless.

1.2.3. Interfaz gráfica con el usuario

Para permitir que el usuario pueda poner en marcha la antena e inicializar los parámetros requeridos para su correcto funcionamiento, es importante que en el punto de monitorización exista una interfaz gráfica que permita cierto control sobre la antena. Para llevar a cabo esta tarea, se implementará un servidor web en la antena que ofrezca los requisitos mencionados a los clientes que se conecten.

1.2.4. Sistema de rastreo

Ya que la antena va a auto gestionar la localización y seguimiento del UAV, es importante que tanto al inicio como en caso de pérdida de localización del avión

durante el vuelo la antena sea capaz de ponerse a buscar hasta encontrar una señal valida de la posición del mismo. Gracias a esta implementación mejoramos la robustez del sistema.

1.2.5. Monitorización de la telemetría desde más de un punto

En el momento que desvinculamos la antena del punto de monitorización por conexión física, se abre un abanico de posibilidades en las mejoras que podríamos implementar. Lo que se ha considerado de utilidad para mejorar la monitorización del vuelo en futuras aplicaciones es la posibilidad de tener más de un ordenador conectado a la antena vía wireless.

1.2.6. Orientación de la antena en el mapa de vuelo

Otro punto importante que ayuda a mejorar la monitorización del avión es el hecho de visualizar, además del propio UAV, la posición y orientación de la antena en el mapa de vuelo en tiempo real. Gracias a esto conseguimos hacer mucho más intuitivo el seguimiento ya que se visualiza en todo momento si la antena apunta correctamente al UAV como se observa en la Figura 1.3.

Figura 3. Visualización de la antena y el UAV en el mapa de vuelo

1.3. Metodología para el desarrollo experimental

Durante la elaboración del proyecto se pueden diferenciar una serie de etapas que se desarrollaron de forma secuencial:

1) En primer lugar está la fase de documentación y aprendizaje de las herramientas básicas que se iban a utilizar. Se leyó gran cantidad de información, tantos de libros como principalmente de la web, sobre la implementación del código y de las funciones que nos serían de utilidad para el desarrollo de los objetivos propuestos. Con ello se ampliaron los conocimientos del lenguaje de programación C necesarios para el entorno de trabajo. También se leyó un libro de iniciación al lenguaje de programación Python que sirvió para establecer los conocimientos básicos necesarios para nuestra aplicación.

2) A continuación, con los conceptos básicos asimilados se procedió a la programación en C de una primera aproximación de lo que sería una solución al problema planteado. Con esta primera versión del software se comprobó la funcionalidad del programa y se procedió a realizar la primera prueba de autonomía de la antena. A pesar de que está primera solución cumplía los requisitos básicos planteados al inicio del proyecto, tenía poca robustez y era necesario implementar mejoras y cambiar gran parte de la estructura del programa. Esta segunda etapa fue la que más esfuerzo y tiempo supuso ya que es en la que se adopta la primera solución válida a partir de los conocimientos adquiridos durante toda la primera fase.

3) Tras una reestructuración en el esqueleto del programa y mejoras importantes a nivel de programación, se obtuvo un código mucho más robusto y modular que sincronizaba varios hilos de ejecución independientes para su correcto funcionamiento. Con estas mejoras se procedió a una segunda prueba en donde se comprobó el correcto funcionamiento de los nuevos cambios software, aunque aún seguían existiendo partes a mejorar en el hardware que imposibilitaban la obtención de los resultados esperados. Esta fase fue la más eficiente del proyecto ya que no solo fue la que más nivel de productividad

aporto, sino que además se consiguió realizar en mucho menor tiempo que la segunda fase gracias a la familiarización que se tenía con el entorno de trabajo y con el lenguaje de programación en sí.

4) Finalmente, se dieron los últimos retoques del software y se solucionaron algunos problemas encontrados en el hardware antes de llevar a cabo la tercera prueba del sistema autónomo. Con este último test se comprobó el correcto funcionamiento autónomo de la antena a la hora de localizar y perseguir al ordenador a bordo del UAV en campo abierto como se había proyectado.

1.4. Estructura del documento

- Capítulo 1 – Introducción y objetivos: El capítulo actual. Se realiza una introducción al Proyecto y se hace una breve referencia a la infraestructura previa existente, se describen los objetivos fundamentales que se pretenden alcanzar, las etapas de desarrollo del sistema a implementar y la estructura del presente documento.

- Capítulo 2 – Estado del arte: En este capítulo se describen el estado de las tecnologías utilizadas en el Proyecto: Vehículos Aéreos No Tripulados (UAVs), concepto de Telemetría, Raspberry Pi como sistema embebido y de la tecnología Wi-Fi. Se realiza una breve introducción histórica de cada punto y de su estado actual.

- Capítulo 3 – Alternativas software utilizadas: Antes de profundizar en el desarrollo de la programación del código, es importante hablar de las alternativas software utilizadas en cuanto al lenguaje de programación empleado. Se hace una introducción al lenguaje C y se explican sus principales ventajas frente a otras alternativas. También se habla del lenguaje de programación Python y el lenguaje de marcas HTML utilizados para la implementación del servidor Web.

- Capítulo 4 – Desarrollo software: Diseño de la solución adoptada en términos de software. Se describe la estructura principal del código desde la función principal (main), los hilos de ejecución independientes y las funciones auxiliares creadas para nuestra aplicación.

- Capítulo 5 – Desarrollo hardware: Descripción de la solución adoptada en términos de hardware. Se exponen y se describen los componentes utilizados, el ensamblaje de los sistemas embebidos y sus interconexiones.

- Capítulo 6 – Pruebas y ensayos: En este apartado se describen los ensayos de funcionamiento realizados. Tanto las pruebas iniciales de comprobación de funcionamiento de los distintos módulos en el laboratorio, los ensayos del sistema completo al aire libre con el ordenador abordo simulando el movimiento del avión, y la prueba final de vuelo.

- Capítulo 7 – Posibles mejoras para futuros proyectos: Se describen posibles alternativas software y hardware que se podrían tomar en el futuro en líneas de mejorar la aplicación.

- Capítulo 8 – Conclusión: Como resumen de la memoria se exponen las conclusiones obtenidas a lo largo de la realización tanto de la parte experimental como de la propia memoria.

- Anexo I – Manual de usuario: Manual elaborado para la puesta en marcha del sistema implementado y para la utilización del sistema autónomo de la estación de control de tierra a nivel de usuario.

- Anexo II – Manual de programación: Se adjunta el código en C completo y se describen las técnicas de programación empleadas en la programación para una posible reconstrucción del proyecto desde cero.

2. ESTADO DEL ARTE

2.1. Introducción a los UAVs

Un vehículo aéreo no tripulado, UAV por las siglas en inglés (Unmanned Aerial Vehicle) es una aeronave que carece de un piloto a bordo. Este dispositivo se pilota, bien desde un ordenador de abordo o bien por un piloto de forma remota desde una estación en tierra u otro vehículo.

Son utilizados fundamentalmente con fines militares, pero también se están usando cada vez más para aplicaciones civiles, como vigilancia, lucha contra el fuego y trabajos de seguridad no militares.

Los globos austriacos

El primer hecho del que se tiene constancia de uso de un vehículo aéreo no tripulado con fines bélicos, tuvo lugar el 22 de agosto de 1849, cuando los austriacos atacaron la ciudad italiana de Venecia, con globos no tripulados cargados con explosivos (1).

Primera Guerra Mundial

La primera aeronave no tripulada se construyó durante y después de la primera guerra mundial. El nombre que se le dio fue '*Aerial Target*', y fue construido en 1916 por *Archibald Montgomery Low* usando sus técnicas de radio control (2). El 12 de septiembre de ese mismo año, el *Hewitt-Sperry Automatic Airplane*, también conocido como 'la bomba voladora', hizo su primer vuelo, demostrando así el concepto de aeronave no tripulada. Estaban destinados para usarse como 'torpedos aéreos', una versión temprana de los modernos misiles de crucero actuales. El control se llevaba a cabo usando giroscopios desarrollados por *Elmer Sperry* de la compañía *Sperry Gyroscope* (3).

Más tarde, en noviembre de 1917, una de estas aeronaves realizó un vuelo para representantes de la armada estadounidense. A raíz de esto, encargaron un proyecto para desarrollar el 'torpedo aéreo', que dio lugar al *Kettering Bug*(**Figura 4**). Fue un gran avance tecnológico, pero no pudo ser aplicado en la guerra porque ésta acabó antes de que la aeronave estuviera completamente desarrollada (4).

Figura 4: Imagen tomada del Kattering Bug

Segunda Guerra Mundial

Reginald Denny and the Radioplane

En 1935 una estrella de cine y entusiasta de los modelos a escala de aeronaves, *Reginald Denny*, mostró un prototipo teledirigido, el *RP-1*, a la armada estadounidense para usarlo en el entrenamiento de las defensas antiaéreas. Poco después, Denny y sus compañeros de trabajo obtuvieron un contrato de la armada estadounidense para desarrollar el *RP-4*, el cual se convirtió en el *Radioplane OQ-2* (**Figura 5**). Llegaron a fabricar cerca de quince mil drones para la armada durante la segunda guerra mundial.

Figura 5: Imagen tomada del OQ-2. Museo de la USAF.

Torpedos aéreos

La Marina de los Estados Unidos, comenzó a experimentar con aviones de radiocontrol en la década de 1930, dando lugar al *Curtiss N2C-2* en 1937 (5). En 1939 la fuerza aérea americana adapto el concepto del *N2C*-2 (5). Tomaron como blancos de entrenamientos, aviones obsoletos bajo el nombre *A-series* que más adelante se les daría la designación *PQ*. Muchos de estos aviones bajo la designación *PQ* fueron usados en la operación *Afrodita* durante la segunda guerra mundial, aunque sin gran éxito.

En 1941 se instaló una cámara de televisión en un drone de ataque y una pantalla de televisión en el avión de control *TG-2*. En 1942 el drone de ataque disparó con éxito un torpedo para destruir un destructor, a una distancia de 20 millas desde el *TG-2* (5).

Guerra Fría

Evolución de los drones como blanco de entrenamiento

El uso de drones como señuelo comenzó en la década de 1950. La primera operación que se conoce fue llevada a cabo por un *ADM-20 Quail* de *McDonnell Douglas*, que fue escoltado por un bombardero *Boeing B-52* para ayudarlo a penetrar el espacio aéreo protegido.

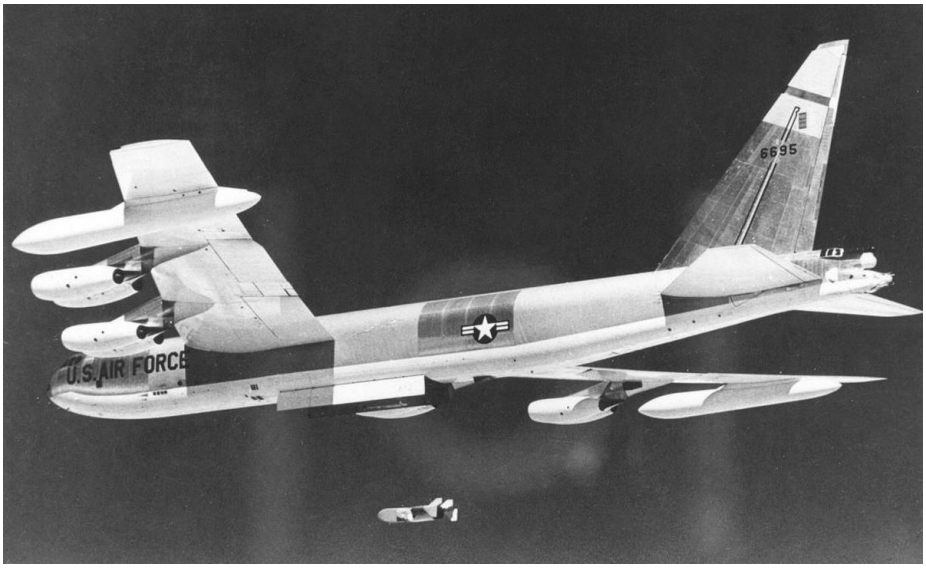

Figura 6: Imagen del ADM-20 Quail y el bombardero B-52.

A finales de 1950, un avión de combate fue capaz de alcanzar una velocidad de vuelo de Mach 2. Northrop diseñó un blanco turbo-propulsado capaz de alcanzar esas velocidades bajo el nombre Q-4, que después pasó a ser llamado *AQM-35*.

Figura 7: Imagen tomada del AQM-35

Plataformas de reconocimiento

A finales de 1950, Los Estados Unidos adquirieron otro dron de reconocimiento, el *Aerojet-General MQM-58 Overseer*. El gran éxito de los drones como blancos de entrenamiento de combate, permitió que se extendiera su uso a otro tipo de misiones. El *Ryan Firebee* fue una buena plataforma para estos experimentos, obteniendo un alto porcentaje de logros en misiones de reconocimiento. A partir de este modelo, surgieron otros como el *Ryan Model 147 Lightning Bug* (**Figura 8**) utilizado por los Estados Unidos para espiar en el norte de Vietnam, la China comunista y Corea del Norte durante la década de 1960 y principio de la década de 1970.

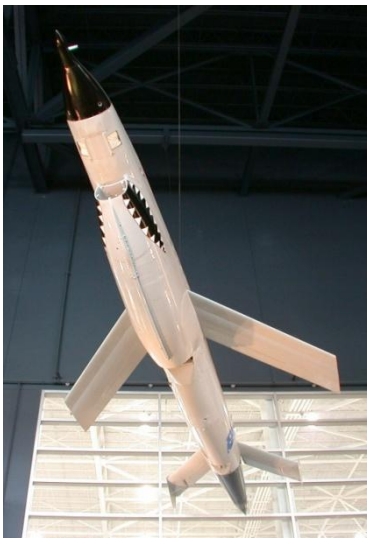

Figura 8: Lightning Bug, Strategic Air & Space Museum, Nebraska

El *Lightning Bug* no fue el único dron de reconocimiento de largo alcance desarrollado en la década de los 60. Los Estados Unidos desarrollaron otros drones de reconocimiento más especializado como el *Ryan Model 154* y el *Lockheed D-21*, los cuales fueron desarrollados en total secreto (6).

Guerra de Vietnam: Plataformas de reconocimiento

A finales de 1959, el único avión espía que poseía los Estados Unidos era el *U-2*. Los satélites espías estaban a más de un año para su puesta en funcionamiento y el *Blackbird SR-71* aún estaba en la en fase de diseño. En este entorno, comenzó a haber preocupación por la publicidad negativa que estaba dando la captura de pilotos estadounidenses en territorio comunista. En mayo de 1960, el miedo de los pilotos se hizo realidad cuando el piloto de *U-2* Francis Gary Powers fue derribado sobre la Unión Soviética.Como era de esperar, esto intensificó el interés en desarrollar un dron que fuera capaz de penetrar en el territorio enemigo y regresar con información valiosa. Pasados tres meses después de este suceso, nació un programa de alto secreto para el desarrollo de un UAV bajo el nombre *Red Wagon* (7).

Justo después de los accidentes con los destructores americanos *USS Maddox* y *Turner Joy*, e incluso antes de la guerra de Vietnam, la Fuerza Aérea de los Estados Unidos (*USAF*) había emitido una orden inmediata para desplegar drones en el sureste asiático. Los primeros en llevar a cabo esta misión fueron los *Ryan 147B* respaldados por los *C-130* (7).

Desde agosto de 1964, hasta el último combate aéreo el 30 de abril de 1975, se habían lanzado un total de 3.435 drones sobre el norte de Vietnam y sus alrededores, con una pérdida de 554 durante el conflicto (7).

Observaciones de la posguerra

La utilidad de los vehículos aéreos no tripulados como plataforma de reconocimiento, fue demostrada en el conflicto vietnamita. Al mismo tiempo se estaban tomando los primeros pasos para su uso activo en combate, tanto en mar como en tierra. Pero no fue hasta la década de 1980 cuando entraron en servicio.

Durante los primeros años, los drones eran lanzados desde un avión, desde un raíl usando cohetes de combustible sólido, o lanzados desde catapultas electromagnéticas, hidráulicas o neumáticas. Sólo unos pocos drones contaban con tren de aterrizaje, y por lo general eran recuperados mediante paracaídas o skis de aterrizaje (7).

Era moderna

UAVs de combate

La postura frente a los UAVs, los cuales eran vistos como poco fiables y como juguetes muy caros, cambió drásticamente con la victoria de la Fuerza Aérea Israelí frente a la Fuerza Aérea Siria en 1982. El uso combinado de UAVs y aviones tripulados, permitieron destruir rápidamente decenas de aviones Sirios con pérdidas mínimos por parte del ejército Israelí. Los aviones no tripulados fueron usados como señuelo, obstáculos y también para el reconocimiento por video a tiempo real (8).

La primera vez que un UAV sin cola fue puesto a prueba fue en 1987, como concepto de prueba para la súper-maniobrabilidad y el uso de la tecnología Stealth (9). Después de varios intentos fallidos, los militares estadounidenses parecían estar formando una flota efectiva de UAVs de combate. El *Predator RQ-1L* (**Figura 9**) de *General Atomic*, fue el primero en ser desplegado en los Balkanes en 1995, en Iraq en 1996 y demostró gran efectividad en la guerra de Iraq y en la de Afganistán.

Figura 9: Predator M/RQ-1

UAVs de gran autonomía

La idea de diseñar un UAV que fuera capaz de mantenerse en vuelo por largos períodos de tiempo, lleva intentando desarrollarse desde hace décadas, pero ha sido en el siglo 21 cuando se ha hecho posible. Actualmente los vehículos no tripulados que operan a grandes altitudes, son completamente operativos y son conocidos como *high-altitude long-endurance (HALE)* (10).

UAVs experimentales alimentados con rayos de energía

A finales de la década de 1950, se estudiaba la posibilidad de utilizar los UAVs como alternativa económica a los satélites para las investigaciones atmosféricas, observación terrestre y marítima y en especial las comunicaciones. Los estudios conceptuales también buscaban una alternativa a los medios de propulsión convencionales, haciendo uso de las microondas o de las células fotovoltaicas.

La compañía *Raytheon* definió lo que hoy se conocería como un UAV impulsado por un rayo de energía, volando a una altitud de 15 kilómetros. Se llevó a cabo una prueba de este concepto en 1964, con una antena de transmisión alimentando a un helicóptero, el cual incorporaba una antena rectificadora (*rectenna*) con cientos de diodos, para convertir la energía del rayo en energía eléctrica.

A pesar de la buena acogida del sistema, no se continuó desarrollando ya que la *rectenna* era muy pesada e ineficiente. Sin embargo, en la década de 1970, la *NASA* se interesó por el éste sistema para su uso en aplicaciones espaciales, y en 1982 publicó un diseño mucho más ligero y económico para la antena rectificadora.

La rectenna de la NASA estaba fabricada con una película fina de plástico, con una antena dipolar y toda la circuitería embebida en la superficie de la antena. En 1987 el Centro de Investigación Canadiense para la Comunicación, utilizo una rectenna mejorada para alimentar un UAV de 5 metros de envergadura y 4,5kg de peso, como parte del proyecto *Stationary High Altitude Relay Platform (SHARP)*. El UAV voló a 150 metros por encima de la rectenna y necesitaba 150 vatios de energía, que se obtenían de un rayo de microondas de entre 6 y 12 kilovatios.

Energía Solar

En la década de 1980, la atención se centró en las aeronaves propulsadas por energía solar. Las células fotovoltaicas no son muy eficientes, y la energía proporcionada por el sol sobre un área es muy modesta. Un avión impulsado por este tipo de energía debe ser ligero, y con motores eléctricos de bajo consumo para que pueda ser capaz de despegar.

En 1980 *DuPont Corporation* respaldo a la compañía *AeroVironment* con la intención de construir un avión tripulado que pudiese volar desde Londres, Paris hasta Inglaterra. El primer prototipo no tuvo mucho éxito, pero dio lugar a un mejor avión, el *Solar Challenger*. Este éxito llevo a *AeroVironment* a plantearse el concepto de UAV alimentado por energía solar. Un avión con estas características puede, en principio, permanecer en el aire de forma indefinida, tanto tiempo como las reservas de energía le permitan volar durante la noche.

En 1983 *AeroVironment* investigó el concepto, con el cual desarrolló el *High Altitude Solar (HALSOL)*. Su primer vuelo fue en junio de ese mismo año. El HALSOL tenía forma de ala volante con una envergadura de 30 metros y una anchura de 2,44 metros. El larguero principal del ala estaba fabricado con tubos de compuesto de fibra de carbono, con costillas hechas de poliestireno y reforzado con abeto americano y Kevlar, todo ello recubierto con una fina capa de plástico Mylar (PET).

El ala estaba construida en cinco segmentos de igual envergadura. Dos góndolas colgaban del elemento central, en el cual se encontraba la carga útil, la telemetría, la electrónica y otros dispositivos como el tren de aterrizaje. El *HALSOL* estaba impulsado por ocho pequeños motores eléctricos dotados con hélices de paso variable. El peso total de la aeronave era de unos185kilogramos, del cual casi un 10% estaba destinado a la carga útil.

El *HALSOL* voló nueve veces a lo largo del verano de1983, en la base secreta de *Groom Lake* en el estado de Nevada. Los vuelos se llevaron a cabo utilizando control por radio y energía suministradas por baterías, ya que aún no había sido equipado con células fotovoltaicas. Se llegó a la conclusión de que la aerodinámica de la aeronave era

eficiente, pero la tecnología de células fotovoltaicas y la de almacenamiento no estaban lo suficientemente desarrolladas para que el diseño fuera llevado a cabo con total éxito.

Amber

En 1984, DARPA otorgó un contrato de 40 millones dólares a *Leading Systems Incorporated* (LSI) de Irvine, California, para construir un UAV llamado *Amber* (**Figura 10**). Iba a ser utilizado para el reconocimiento fotográfico, misiones *ELINT*, o como un misil de crucero. El ejército estadounidense, la Armada y la Marina estaban interesados, pero DARPA finalmente cedió el control a la Armada.

Amber fue diseñado por un equipo dirigido por Abraham Karem de *LSI*. El aparato tenía 4,6 metros de largo, 8,54 metros de envergadura y pesaba 335 kilogramos. Estaba dotado de un motor de combustión interna de cuatro cilindros, refrigerado por agua y que proporciona 65 CV para mover una hélice en la cola del avión. El *HALSOL* fue descartado por el *Amber* en su última misión.

Figura 10: UAV Amber

Amber tenía una cola invertida en V, lo que proporciona protección a la hélice durante el despegue y el aterrizaje. El fuselaje estaba fabricado de plástico y materiales compuestos, en su mayoría de Kevlar. El UAV contaba con un tren de aterrizaje de tres ruedas retráctil y su autonomía era de más de 38 horas.

El contrato inicial establecía tres prototipos *Amber Basic A-45* y tres prototipos *B-45* de reconocimiento. Los vuelos iniciales se llevaron a cabo en noviembre de 1986, y los vuelos de larga duración al año siguiente. Hasta ese momento, el desarrollo del *Amber* era alto secreto, pero en 1987 los detalles del programa se dieron a conocer.

El *Amber* sólo era uno de tantos programas en activo de la época, para el desarrollo de aeronaves no tripuladas y el Congreso estadounidense se impacientaba por el esfuerzo y la confusión que suponía tener varios programas dedicados al mismo fin. El Congreso ordenó la unificación de los programas en 1987, congelando la financiación hasta junio de 1988, cuando la *Joint Program Office* para el desarrollo de UAVs fue creada y establecida. El programa *Amber* se mantuvo en espera hasta la unificación, dando lugar al *Amber I* que voló por primera vez en octubre de 1989. Siete *Amber I* fueron fabricados y usados en las evaluaciones junto al *Amber Basic* hasta el año 1990. Sin embargo, se hicieron recortes en la financiación y el programa *Amber* fue cancelado. La compañía LSI se enfrentaba a la banca rota y fue comprada por General Atomics en el año 1991, que más tarde desarrollaría el proyecto *Amber* en una nueva plataforma, el *MQ-1 Predator* (**Figura 9**) (11).

Uso civil

La agencia de Aduanas de EE.UU. y Protección Fronteriza ha experimentado con varios modelos de vehículos aéreos no tripulados, y ha comenzado a comprar una flota de desarmados *MQ-9 Reapers* para vigilar la frontera con México (12).

Figura 11: MQ-9 Reapers

El 20 de marzo de 2007, un UAV de la agencia de protección de fronteras no tripulado UAV detectó a seis sospechosos extranjeros y guio a los agentes hasta ellos, incluido el mexicano Leopoldo López-Aparicio, que era buscado en el estado de Washington por cargos de violación en tercer grado de un niño (13).

El 18 de mayo de 2006, la *Federal Aviation Administration* (*FAA*) de los Estados Unidos, emitió un certificado de autorización que permitía a la aeronave *M/RQ-1* y *M/RQ-9*, ser utilizada en el espacio aéreo civil estadounidense para buscar sobrevivientes en cualquier tipo de desastre. La solicitud se había realizado en 2005 para usar una aeronave en las operaciones de búsqueda y rescate tras el huracán Katrina, pero no llegó a utilizarse debido a que no existía la autorización por parte de la *FAA* (14).

UAVs asequibles

Las tecnologías dedicadas al uso de los vehículos aéreos no tripulados siempre habían estado limitadas casi exclusivamente a uso militar debido a su alto coste. Pero desde hace algunos años, con la caída de los costos de estas tecnologías y los equipos de monitoreo, se ha facilitado el acceso a grupos con menos recursos. A partir de 2004, la milicia chiíta libanesa puso en activo el *Misrad-1*, con el objetivo de dotarlo de armamento y utilizarlo en ataques fronterizos contra Israel (15).

2.2. Introducción a la telemetría

Un sistema de telemetría es un caso partículas de sistema de comunicación cuyo objetivo es el recoger datos de un lugar de difícil accesibilidad física para llevarlos a un punto en donde pueden ser evaluados. Habitualmente los sistemas de telemetría se utilizan para obtener información de vehículos móviles tales como coches, aviones o misiles desde los cuales se generan parámetros tales como su posición, orientación, inercia gracias a sensores solidarios al vehículo (16).

2.2.1. Diseño general de un sistema de telemetría

El diseño general de un sistema de telemetría se compone de una serie de etapas desde que los datos a analizar son generados en el vehículo hasta que son procesados en su punto de destino (16):

1. Sistema de generación de datos.
2. Sistema de multiplexado.
3. Modulador y Antena transmisora.
4. Canal de transmisión.
5. Antena receptora y Demodulador.
6. Sistema de de-multiplexado.
7. Sistema de procesado de datos.

En primer lugar los datos deben ser recogidos por los distintos sensores o transductores que convierten variables físicas en señales eléctricas. Estas señales eléctricas son generalmente muy pequeñas y por tanto deben ser amplificadas antes de ser enviadas al bloque multiplexor.

En la segunda etapa, los distintos datos separados deben ser combinados para poder ser enviados por un mismo canal de transmisión. Esta tarea es realizada por un sistema de multiplexación, el cuál comprime los datos FDM (frecuency división multiplexing) o TDM (time división multiplexing) en función de si los datos estaban separados en el dominio de la frecuencia o del tiempo respectivamente.

El tercer subsistema formado por el modulador y la antena transmisora se encargan de modular los datos multiplexados en una señal portadora para así poder ser transmitidos a través de la antena.

El cuarto bloque consiste es el propio canal de transmisión que normalmente es el aire y por el cual la señal portadora de la información es radiada desde la antena transmisora hasta la antena receptora. Para comprobar la calidad de transmisión es necesario conocer parámetros como la ganancia de la antena transmisora, la atenuación de la señal en el medio que en este caso sería el aire, las características de la antena receptora y la potencia efectiva transmitida. También es importante denotar que la frecuencia de la señal portadora a la que se transmite los datos, debe ser compatible con las antenas transmisora y receptora. Esta frecuencia generalmente viene determinada por las dimensiones de la antena y la longitud de onda de la señal portadora, las cuales deben estar en el mismo rango de tamaño.

La señal portadora es recogida en la antena receptora y demodulada en la quinta etapa, obteniendo así la señal con los datos originales. Está señal nueva sin la portadora será demultiplexada en el sexto subsistema, como proceso contrario a la compresión de los datos en la etapa dos, con el objetivo de obtener los datos separados para que puedan ser procesados y evaluados en la etapa final.

Como esquema grafico del proceso explicado, en la siguiente figura se muestra un ejemplo de un sistema de telemetría de un misil en donde se identifican cada una de las siete etapas descritas anteriormente (16).

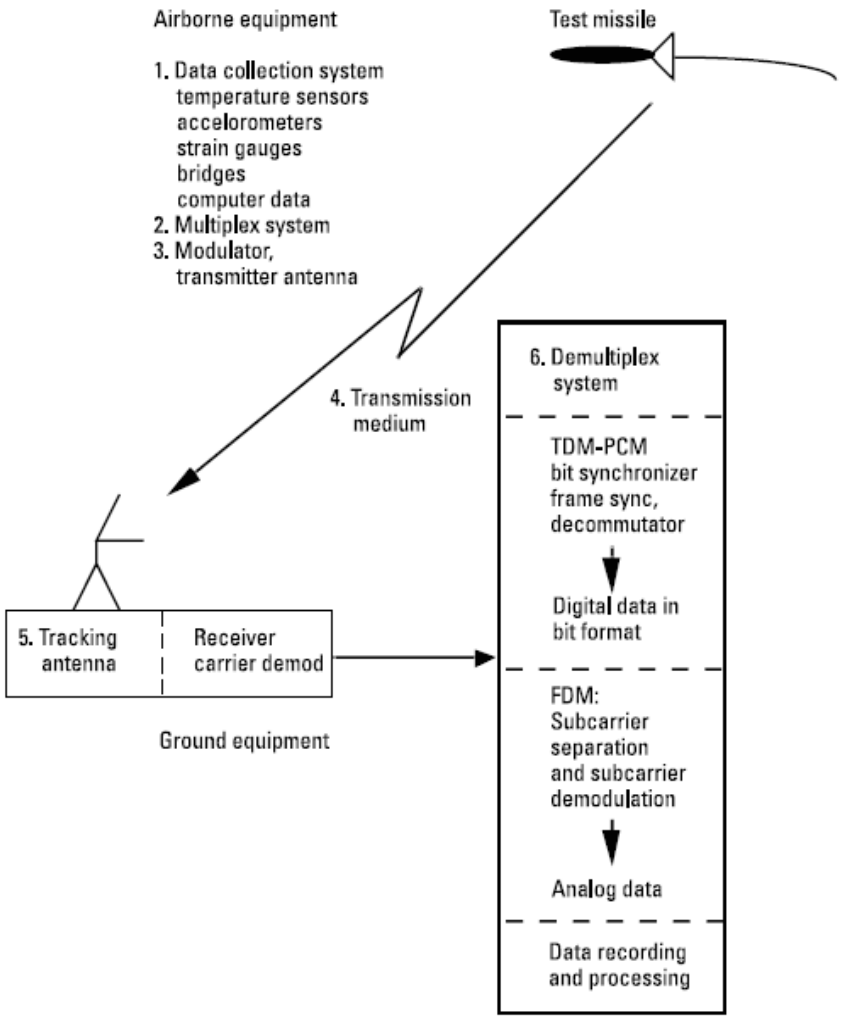

Figura 12. Ejemplo de las etapas en el sistema de telemetría de un misil

2.2.2. Historia de la telemetría

La telemetría de información por cable tiene sus orígenes en el siglo XIX, durante el cual se fueron desarrollando distintas aplicaciones. Uno de los primeros circuitos de transmisión de datos fue desarrollado en 1845 entre el "Winter Palace" del emperador de Rusia y el cuartel general del ejército. En 1874, ingenieros franceses construyeron un sistema de telemetría capaz de enviar datos en tiempo real sobre la meteorología y los sensores profundidad de la nueve del Mont Blanc hasta Paris. Más tarde, en 1901 fue patentado por el inventor americano C. Michalke un circuito de envío sincronizado de parámetros de rotación a distancia cuya principal utilidad era medir el ángulo de maquinas rotativa como por ejemplo la plataforma de una antena. En 1906 fueron construidas en el observatorio Pulkovo en Rusia, una serie estaciones sísmicas con telemetría para recoger datos sobre terremotos y movimientos sísmicos. Por otro lado,

en el Canal de Panamá terminado entre 1913 y 1914 también fueron implementados sistemas de telemetría para monitorizar los niveles de agua y las esclusas (17).

Hasta ahora, las aplicaciones mencionadas eran a través de circuitos y cableados. Los primeros sistemas telemétricos “wireless” o “sin cable” fueron desarrollados en 1930 por Robert Bureau en Francia y Pavel Molchanov en Rusia para aplicaciones en radio sondas. El sistema de Molchanov consistía en modular las medidas de temperatura y presión y transmitiéndolas por código Morse vía wireless. También fue implementado un sistema llamado “Messina” de multiplexado de señales de radio en la nave V-2 alemana, pero era poco fiable y tanto en EEUU como en la URSS fueron rápidamente reemplazadas por sistemas mejores basados en la modulación por posición de pulso (PPM) (18).

Figura 13. Nave alemana V-2 en el Museo Peenemünde (Alemania).

Los primeros misiles soviéticos y sistemas de telemetría en el espacio fueron desarrollados a finales de la década de 1940, los cuales también utilizaban la modulación PPM o la modulación por duración de pulso (PDM). Los primeros trabajos estadounidenses también empleaban sistemas similares que más tarde fueron sustituidos por la modulación por código de pulso (PCM), un ejemplo fue la exploración a Marte con la sonda Mariner 4 en 1964 (19).

Figura 14. Fotografía de la sonda Mariner 4

Más tarde, sondas interplanetarias soviéticas ya comenzaron a utilizar sistemas de radio redundantes, transmitiendo por modulación PCM en una banda de decímetros y por modulación PPM en una banda de centímetros (20).

2.3. Introducción a los sistemas embebidos

Un sistema embebido se define como un sistema computacional que ha sido diseñado para llevar a cabo tareas generalmente en el ámbitos de sistemas de tiempo real. En nuestro caso se precisa de este sistema empotrado para poder programar toda la gestión tanto de la telemetría como del control de la antena.

Por ello, una de las bases hardware de nuestro proyecto es una placa computadora de bajo coste Raspberry Pi la cual tiene las dimensiones aproximadamente de una tarjeta de crédito.

Uno de los principales objetivos del proyecto es poder tener la antena y su controlador como un solo bloque compacto. De ahí la necesidad de que nuestro sistema embebido tenga un diseño lo suficientemente reducido como es el caso de la Raspberry, para que pudiese ajustar en la carcasa cerrada solidaria a la antena.

En la siguiente imagen se muestra la Raspberry Pi utilizada:

Figura 15. Placa computadora Raspberry Pi Modelo B

Historia y características de la Raspberry Pi

Fue lanzada al mercado en Febrero de 2012 (21) por la Fundación Raspberry Pi, una asociación inglesa que apostaba por desarrollar computadores programables a un precio accesible incluso para personas en países en desarrollo, y que a su vez tuviese las características de un ordenador competente a día de hoy.

Este dispositivo trabaja con el sistema operativo gratuito Raspbian, basado en Debian de Linux y optimizado para trabajar con la Raspberry Pi. Sus características principales incluyen un procesador ARM a 700MHz, equiparable a un antiguo Pentium III pero con propiedades graficas de un ordenador sobremesa actual. Contiene una memoria RAM de 512MB y carece de disco duro o unidad de estado sólido ya que su precisa de una tarjeta SD para el almacenamiento permanente del sistema operativo y de la memoria de datos (22).

La Raspberry Pi es alimentada con 5V con un adaptador gracias a una entrada micro USB o bien por los pines de la cabecera GPIO que posee. Actualmente existen dos modelos en el mercado en función de sus características y por lo tanto de su precio. Para nuestra aplicación, utilizamos el modelo B el cuál posee dos entradas USB 2.0, una entrada Ethernet RJ45, salida de audio bien por HDMI o por conector de 3.5mm, salida de video tanto por conector RCA como por HDMI, pines GPIO para periféricos de bajo nivel como UART o SPI y es alimentada con un adaptador de 5 V gracias a una entrada micro USB o bien por los pines de la cabecera GPIO que posee (22).

En el siguiente diagrama se muestra el diseño del modelo B de Raspberry utilizado y se puede identificar con más detalle cada una de las partes que la componen:

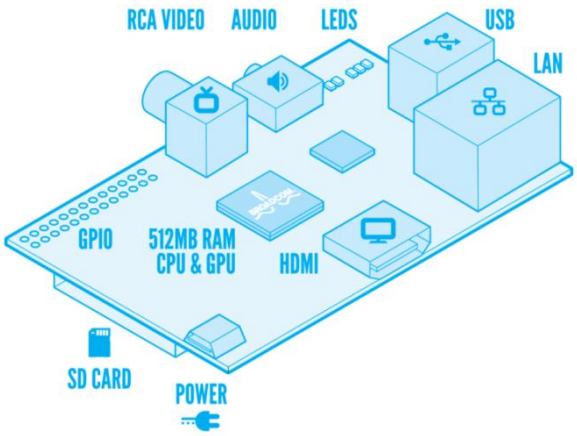

Figura 16. Componentes de la Raspberry Pi Modelo B

A parte de las características técnicas ya mencionadas. A continuación se muestra una tabla con las especificaciones técnicas y dimensionales más en detalle (22):

Tabla 1. Características técnicas de ambos modelos de Raspberry Pi

	Modelo A	Modelo B
SoC	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)	
CPU	ARM1176JZF-S a 700 MHz (familia ARM11)	
GPU	Broadcom VideoCore IV,59 , OpenGL ES 2.0, -2 y VC-1 (con licencia), 57 1080p30 H.264/MPEG-4 AVC3	
Memoria (SDRAM)	256 MB	512 MB
USB 2.0	1	2 (vía hub USB integrado)
Salidas de video	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD	
Salidas de audio	Conector de 3.5 mm, HDMI	
Almacenamiento	SD / MMC / ranura para SDIO	
Conectividad de red	Ninguna	10/100 Ethernet (RJ-45) via hub USB
Otros periféricos	8 x GPIO, SPI, I^2C, UART	
Consumo	500 mA (2.5 W)	700 mA (3.5 W)
Alimentación	5 V vía Micro USB o GPIO header	
Dimensiones	85.60mm × 53.98mm (3.370 × 2.125 inch)	
S. O. soportados	Debian, Fedora, Arch Linux,Slackware Linux, RISC OS	
Precio (solo placa)	25$US	35$US

Como resumen, se puede decir que nos hemos apoyado en la Raspberry Pi como solución hardware de nuestro sistema embebido por tres puntos características importantes:

- Bajo coste: La adquisición del modelo B por un precio que no supera los 35€ es una ventaja potencial frente al coste que podría conllevar la utilización cualquier otro computador del mercado.
- Pequeño tamaño: Sus dimensiones reducidas y su peso son muy importantes a la hora de conseguir un ensamblaje compacto junto con la antena.
- Capacidad de computación: Debido a que precisábamos de un computador única y exclusivamente para arrancar y gestionar las funciones de control y telemetría de la antena, aunque la Raspberry tenga una capacidad limitada, es suficiente para esta dicha tarea.

2.4. Introducción a la tecnología Wifi.

Una parte muy importante de nuestro trabajo se basa en la transmisión de información por medios no físicos. Para ello se recurre a la tecnología Wireless utilizando la banda de frecuencia libre en torno a los 2,4GHz que comúnmente conocemos con la banda de Wifi.

Como su propio nombre indica, la comunicación wireless consiste en la transmisión de información entre dos o más puntos que no están conectados por medio de un cable o conductor eléctrico. La tecnología wireless más común utiliza comunicación wireless electromagnética tales como las ondas de radio, y permite comunicaciones de largo rango que no serian prácticas a la hora de implementarlas mediante conexión de cable.

2.4.1. Historia de las primeras aplicaciones sin cables.

La primera conversación telefónica sin cables ocurrió en 1880 cuando Alexander Graham Bell y Charles Sumner Tainter inventaron y patentaros el fotófono, un teléfono que transportaba el audio sin cables gracias a la modulación de haces de luz (23). Por aquel entonces cuando aun no existían láseres para hacer posible esta comulación, no había una manera práctica de llevarla acabo dado que se dependía de la luz solar y de una buena climatología. Además de estos inconvenientes, era necesaria también una clara línea visual entre el transmisor y el rector. Iba a ser varias décadas después cuando el principio de funcionamiento del fotófono fuera a ser aplicado en comunicación militar y más tarde en comunicaciones de fibra óptica.

A continuación se ilustra una de las primeras aproximaciones de un fotófono transmisor y un fotófono receptor de audio sin cables por modulación de haz de luz. En la siguiente figura se puede observar el camino de la luz solar antes y después de ser modulada.

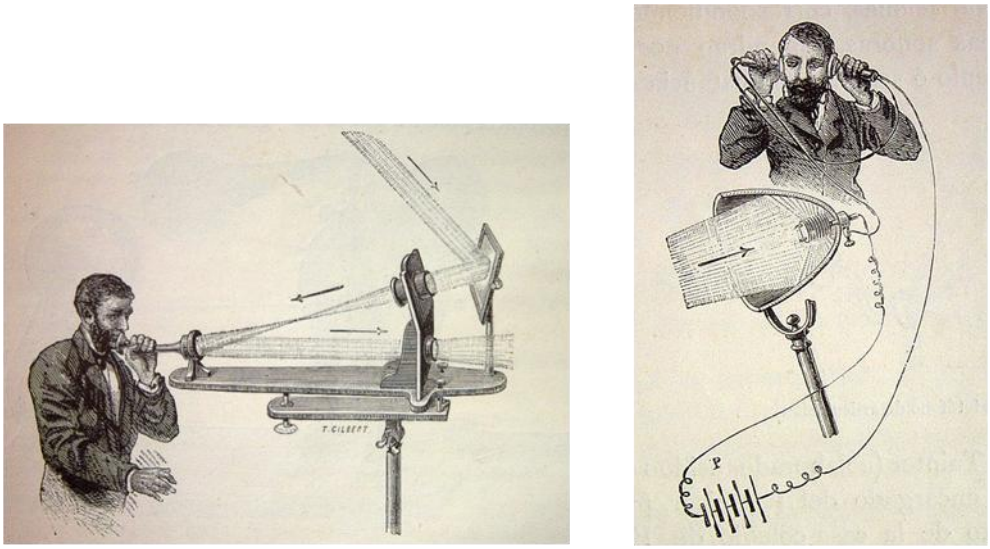

Figura 17. Fotófono transmisor (izquierda) y receptor (derecha)

En 1888 fue demostrada la existencia de ondas electromagnéticas por Heinrich Hertz, la base de la tecnología wireless. Aunque la teoría de las ondas electromagnéticas ya fue predicha por James Clerk Maxwell y Michael Faraday, Hertz demostró que las ondas electromagnéticas viajaban por el espacio en líneas rectas, y que por tanto podían ser transmitidas y recibidas por aparatos fabricados para tal propósito (24). Hertz no continúo con la experimentación de aquellos aparatos transmisores y receptores, y fue Jagadish Chandra Bose quien por ese tiempo desarrollo el primer dispositivo detector de wireless aportando una gran ayuda al conocimiento de las ondas electromagnéticas milimétricas. Aplicaciones prácticas de comunicación de radio wireless y control remoto por radio fueron implementadas más tarde por inventores como Nikola Tesla.

2.4.2. Historia de la comunicación Wifi.

A continuación se trata de explicar la evolución del término Wifi desde sus inicios. La marca Wi-Fi pertenece a la asociación comercial The Wi-Fi Alliance y lo define como cualquier producto “wireless local area network” (WLAN) que esté basado en el protocolo estándar IEEE 802.11 (25). Sin embargo, la mayoría de los WLANs hoy en día están basados en estos estándares, por lo que el término Wifi es usado prácticamente como sinónimo de WLAN.

Todo comienza en 1985, cuando la Federal Communications Commission americana (FCC), tomó la decisión de abrir varias bandas del espectro wireless con el fin de permitir su uso libre sin licencia del gobierno (26). Fueron cogidos tres "trozos" de espectro de las bandas utilizadas en la industria y en la ciencia y se dejaron libres para empresas y emprendedores. Estas tres frecuencias eran 900MHz, 2.4GHz y 5.8GHz y se catalogaron como las "bandas desecho" o "basura" ya que también eran utilizadas por aparatos con fines distintos a la comunicación, como por ejemplo el horno microondas que utiliza la radio frecuencia para calentar los alimentos.

Para hacer posible el uso de estas bandas de frecuencia para la comunicación, la FCC puso como condición que cualquier aparato que operase a estas frecuencias tendría que esquivar las interferencias de otros equipos. Esto se conseguiría utilizando la tecnología de "propagar el espectro" originalmente desarrollada para el uso militar que consiste en propagar la señal de radio a un amplio radio de frecuencias en lugar de una frecuencia puntual. De este modo se consigue que la señal sea difícil de interceptar por otros equipos no deseados, y a la vez menos susceptible a las interferencias.

A lo que más importancia se le dio tras establecer estas bandas de frecuencia sin licencia, fue la idea de tener un estándar en el uso de la tecnología. Inicialmente los vendedores de equipos de wireless para redes locales (LANs) como Proxim y Symbol, desarrollaron sus propios dispositivos para operar en estas bandas sin licencia. El problema era que entre los equipos de diferentes firmas no podían comunicarse, por lo que inspirados en la tecnología de acceso Ethernet, varias compañías se dieron cuenta de que una tecnología común tendría más sentido. Los usuarios tenderían a adquirir aparatos que no estuvieran ligados única y exclusivamente a trabajar y comunicarse con otros equipos de la misma compañía.

En 1988 la compañía NCR Corporation (26) quiso comenzar a usar el espectro sin licencia para aplicar la tecnología wireless a las maquinas de pago por tarjeta. Era necesario basarse en un estándar como el que ya existente estándar de Ethernet IEEE 802.3 (Institute of Electrical and Electronics Engineers), de modo que se estableció el nuevo estándar 802.11 para la comunicación por Wifi. En el mercado fragmentado por

aquel entonces, se tardo bastante tiempo en que este estándar fuese aceptado por la mayoría de los miembros del comité, y finalmente en 1997 se llego al acuerdo de unas bases. Se estableció una velocidad de transferencia de datos de dos megabits por segundo y para ello se podían usar dos modalidades, la primera evitaba las interferencias de otras señales mediante saltos entre radio frecuencias y la segunda propagaba la señal en un rango amplio de frecuencias como se describió anteriormente.

Los ingenieros empezaron a trabajar rápidamente sobre estas nuevas bases, y aparecieron dos variantes, la 802.11b que operaba en la banda de 2.4GHz y la 802.11a que operaba a 5.8GHz ratificadas en Diciembre de 1999 y en Enero del 2000 respectivamente (26). El estándar más divulgado fue el 802.11b con el cual un grupo de seis compañías actualmente conocidas como Intersil, 3Com, Nokia, Cisco, Motorola y Alcatel-Lucent crearon la alianza Wireless Ethernet Compatibility Alliance (WECA) que en 2002 pasó a llamarse Wi-Fi Alliance, la cual da nombre a lo que hoy en día conocemos como tecnología Wi-Fi.

3. ALTERNATIVAS SOFTWARE UTILIZADAS

3.1. Ventajas de la programación en C

Antes de profundizar en las ventajas y desventajas de la utilización del lenguaje C como herramienta de trabajo para el desarrollo software de este proyecto, se expondrá una breve introducción histórica del este lenguaje a fin de colocarnos en el contexto de la aparición de este lenguaje.

3.1.1. Introducción histórica del lenguaje C

Hacia los años 1972 y 1973, el lenguaje C fue creado por un equipo de programadores perteneciente a los antiguos Laboratorios Bell de AT&T.

Fue Dennis Ritchie quien diseño e implemento el primer compilador de este lenguaje en un antiguo computador PDP-11. Por aquel entonces, C se basó en dos lenguajes que hoy en día están prácticamente desaparecidos. Por un lado “BCPL” y por otro “B”. Este último fue escrito en 1970 para un sistema UNIX en un PDP-7 (27).

El nombre oficial de lo que hoy en día se conoce como lenguaje C era “K&R C” y proviene de los autores del libro “The C Programming Language”, Brian Kernigham y Dennis Ritchie. Este libro fue durante muchos años la principal referencia para los programadores de C, hasta que en torno a 1989 se adoptó una versión mejorada del lenguaje por el American National Standars Institute (ANSI) (27), conocida hoy en día como “ANSI C” o “C89”. Esta versión contiene una importante revisión de la sintaxis como es la estandarización parcial de las librerías del sistema, y especialmente la revisión del uso de funciones la cuál fue tomada de su progenitor C++.

Tras posteriores revisiones, la última versión conocida del lenguaje C es “C99” la cual no es actualmente soportada en forma completa por la mayoría de compiladores. Por este motivo hoy en día la gran mayoría de los programadores siguen utilizando la versión “C89”.

3.1.2. Criterios más significativos de comparación

A pesar de que el análisis de las características ventajosas de un lenguaje de programación no es una tarea fácil, en el libro "Lenguajes de Programación" del autor Allen B. Tucker, Jr. (28) se describen una serie de criterios que nos ayudan a evaluar y puntuar los puntos fuertes y débiles del lenguaje C:

1. Expresividad: Facilidad del lenguaje para expresar los algoritmos
2. Modularidad: permitir el desarrollo de componentes independientemente
3. Transportabilidad/Portabilidad
4. Eficiencia
5. Pedagogía: Facilidad de aprendizaje y enseñanza
6. Generalidad: Aplicabilidad, Uso

Desde el punto de vista de la **expresividad**, podemos decir que el lenguaje C era distinguido como muy expresivo y bastante económico en cuanto a que su cantidad de palabras clave es bastante reducida, también por la versatilidad que proporcionan operadores como los punteros. Aunque por otro lado en la actualidad es más frecuente el uso de estructuras de programación cada vez más complejas, lo que convierte al lenguaje C en más abstracto frente a otros lenguajes. En la programación requerida para nuestro proyecto no es muy abundante el uso de las variables de tipo estructura, por lo que el lenguaje C no supone una importante abstracción en la sintaxis a la hora de programar.

En cuanto a la **modularidad**, el lenguaje C permite desarrollar componentes de manera independiente y que en ocasiones interactúen entre ellos. Este aspecto es bastante ventajoso en nuestra aplicación ya que se requieren de varios hilos de ejecución trabajando a la vez con diferentes tiempos de ciclo, pero que a su vez interactúan entre ellos por medio de variables globales compartidas.

Desde el punto de vista de la **transportabilidad** y la **portabilidad**, tradicionalmente el lenguaje C se ha proporcionado como parte de la distribución del sistema operativo Unix lo que implica su reconocimiento como uno de los lenguajes de mayor difusión y portabilidad de la historia. Además, su estandarización fue cuidadosamente elaborada

lo que le convierte en uno de los primeros éxitos de los estándares abiertos. La versión C99 sin embargo no es soportada en un amplio número de compiladores, pero las versiones C89 y C90 son universalmente portables. Por este motivo y con la intención de que nuestro código pudiese ser leído y compilado en otras maquinas a parte del sistema embebido donde está programado, el lenguaje C es una buena solución.

A la hora de hablar de **eficiencia**, nos referimos a la velocidad con la que el lenguaje logra llevar a cabo diversas tareas. En cualquier maquina, los programas son ejecutados en la unidad central de procesamiento conocida como CPU, las cuales solo entienden el "lenguaje maquina". Este código maquina consiste en operaciones de bajo nivel del tipo sumar dos números o escribir bytes en la unidad de memoria. Es por esto que los lenguajes de programación deben ser convertidos o compilados a este código máquina para poder ser ejecutados por el procesador. En el caso del lenguaje C, el programador llega a estar cerca de la programación en "lenguaje maquina" debido a la simplicidad de las estructuras de las que este se compone. Por tanto al haber una correlación tan directa entre lo que el programador escribe y lo que la CPU comprende, el código maquina resultante es más eficiente gracias a su simplicidad. Este es uno de los principales puntos positivos a la hora de elegir el lenguaje C, ya que la aplicación de nuestro proyecto exige una cantidad elevada de operaciones lógicas y matemáticas por cada ciclo de ejecución las cuales consumirían mayores tiempos de CPU si utilizásemos lenguajes de más alto nivel como por ejemplo Java. Otro punto a destacar de tener ante nosotros un lenguaje de nivel medio es la posibilidad de operar a nivel de "byte" lo cual es imprescindible para nosotros a la hora de tratar las tramas con los datos de la telemetría. Además de ello, C nos permite tener un control preciso de los tiempos de ciclo de ejecución lo cual es muy importante a la hora de sincronizar los hilos de ejecución que interactúan en nuestro programa.

Como punto negativo para el lenguaje utilizado habría que destacar la **pedagogía** o la facilidad para aprenderlos. Y es que el lenguaje C fue creado para ser eficiente como acabamos de comentar, y no para que fuese fácil de comprender y utilizar. Este es uno de los principales motivos por el cual no es un lenguaje que haya tenido una rápida adopción y se utilice especialmente para aplicaciones de más bajo nivel.

Por último, el concepto de **generalidad** define la capacidad de un lenguaje para poder abordar prácticamente cualquier clase de problema. En nuestro caso, el lenguaje C cumple con esta característica y fue también uno de los puntos clave a la hora de abordar un problema del que a priori no se conocían las dimensiones ni los recursos que podría requerir. Por este motivo se precisaba de un lenguaje que contase con un amplio espectro para llevar a cabo el proyecto.

Para presentar más claramente todo lo anterior se presenta a continuación una tabla donde se resumen los puntos fuertes y débiles del lenguaje C para nuestro proyecto:

Tabla 2. Resumen de la ventajas e inconvenientes del lenguaje escogido

Criterios	Como lenguaje	Para nuestro proyecto
Expresividad	Media	No relevante debido al poco uso de variables de tipo estructura
Modularidad	Buena	Ventajosa a la hora de programar tareas de ejecución independientes
Portabilidad	Muy buena	Interesante en el caso de querer compilar nuestro código en otro sistema
Eficiencia	Excelente	Imprescindible para las operaciones lógicas y el tratamiento de tramas de la telemetría
Pedagogía	Baja	Punto negativo a la hora de aprender nuevas técnicas de este lenguaje
Generalidad	Buena	Muy importante debido al desconocimiento a priori de la envergadura del problema planteado

3.2. Otros lenguajes utilizados

A parte de la implementación del código en lenguaje C, se ha hecho uso de otros lenguajes para aplicación del servidor web. A continuación haremos una breve descripción de los otros dos lenguajes utilizados.

3.2.1. Lenguaje Python

Python es un lenguaje de alto nivel y de uso general cuya principal filosofía es conseguir facilidad en su lectura y legibilidad. Es un lenguaje que permite a los programadores expresar conceptos en menos líneas que en otros lenguajes como por ejemplo C, y que además permite una programación orientada a objetos y estilos de programación funcional que proporciona un sistema de tipo dinámico y una gestión automática de la memoria a demás de una amplia y comprensible librería.

3.2.1.1. Breve introducción histórica

Los inicios de este lenguaje se remontan a los años 80 y su definitiva implementación no llego hasta Diciembre de 1989 por su autor Guido van Rossum en el "Centrum Wiskunde & Informatica" (CWI) en los Paises Bajos. Guido van Rossum trabajó en la implementación del lenguaje de programación ABC a principio de los 80 y estuvo en ese proyecto hasta 1986 cuando, dentro del mismo CWI, cambio a trabajar en el desarrollo de un sistema operativo llamado Amoeba. Gracias al gran grado de libertad que Guido tenía en este nuevo proyecto y a la necesidad de crear un lenguaje de programación basado en scripts compatible con este sistema operativo, comenzó a trabajar en su propio mini-proyecto. Partiendo de la experiencia y en las frustraciones de su trabajo con ABC, decidió diseñar un lenguaje simple que poseyera lo mejor de este pero sin sus problemas.

Comenzó creando una maquina virtual simple, un analizador sintáctico (parser), una maquina simple de ejecución y una propia versión del lenguaje basada en las partes que más le interesaban de ABC. El punto más negativo que Guido quiso cambiar de este lenguaje era la poca extensibilidad de este frente a otros sistemas operativos o aplicaciones de más bajo nivel. Por ello hizo que Python fuera capaz de ser usado en

otras plataformas empezando Amoeba en el que actualmente trabajaban, pero también en UNIX e incluso en Windows y Macintosh (29).

3.2.1.2. Ventajas de Python

Python es un potente lenguaje de programación utilizado en una amplia variedad de dominios. Normalmente es comparado con lenguajes como Perl, Ruby, Scheme o Java y algunas de sus propiedades más distinguidas son:

- La claridad y comprensibilidad sintáctica
- La intuitiva orientación a objetos
- La gran capacidad de modularidad
- El alto nivel de las bases de datos dinámicas
- La capacidad de extenderse a módulos escritos en otros lenguajes como C

Permite escribir de forma clara y rápida gracias a su compilador optimizado y a sus librerías de soporte. A diferencia de otros lenguajes, gracias a su simplificada sintaxis es por ejemplo capaz de crear un servidor web con prácticamente tres líneas de código.

Al ser un lenguaje orientado a objetos, permite usar clases ya creadas en sus librerías y modificarlas o heredarlas para poder crear las nuestras propias según los intereses de nuestra aplicación sin tener que partir desde cero.

También es capaz de interactuar con otros lenguajes gracias a su extensibilidad y a la capacidad de implementar librerías. Un ejemplo de extensibilidad es la utilización de librerías Python en el lenguaje Java denominadas Jython e implementadas en la propia maquina virtual de Java. Incluso si es necesario programar una aplicación de más bajo nivel que no fuese posible con las librerías propias de Python, es posible escribir módulos de extensión en lenguaje C o C++ e integrarlos.

Otra de sus características más importantes es que Python está disponible en la gran mayoría de los sistemas operativos como Windows, Linux/Unix, OS/2, Mac, Amiga, entre otros. En la mayoría de los sistemas operativos en donde existe un compilador para C, probablemente esté disponible también una plataforma para arrancar Python.

3.2.1.3. Servidor web

En definitiva, a la hora de realizar el servidor web para nuestra aplicación con la antena partimos de un modelo ya existente programado en Python utilizado en aplicaciones anteriores. Gracias a la extensibilidad del lenguaje, lo editamos de manera que nos permitieses implementar una interfaz en lenguaje HTML y también almacenar parámetros que el usuario introducir gracias a una pequeña base de datos creada para tal función.

3.2.2. Lenguaje HTML

Las siglas HTML vienen del término "Hyper Text Markup Language" o lenguaje de marcado hipertextual. Un lenguaje marcado o lenguaje de marcas es una manera de codificar un documento que incorpora etiquetas que contienen información adicional sobre la estructura del texto o de su presentación. Entre estos tipos de lenguajes, el más extendido es el HTML, base del famoso World Wide Web. A diferencia de los lenguajes de programación (C, C++, Python, Java), los lenguajes de marcas carecen de funciones aritméticas o variables. Están destinados a otras funcionalidades como por ejemplo la creación y diseño de páginas web.

3.2.2.1. Breve introducción histórica

Como ya hemos mencionado, HTML es el lenguaje oficial del conocido acrónico WWW y fue concebido por primera vez en 1990. Es un producto de "Standard Generalized Markup Language" (SGML), dicho estándar está destinado a describir las especificaciones de lenguajes de marcas y en especial a aquellos utilizados en el intercambio de documentos electrónicos, organización de documentos y publicado de documentos. HTML fue originalmente creado para permitir (a aquellos que no estuvieses especializados en SGML) publicar e intercambiar documentos científicos y otros documentos técnicos. Este intercambio está especialmente facilitado gracias a la incorporación de "hyperlinks" o hipervínculos que habilitan links a unos documentos electrónicos dentro de otros. De aquí el nombre de Hypertext Markup Language. Con el tiempo se descubrió que HTML era relativamente fácil de aprender y fácil de integrar en numerosas aplicaciones, por lo que con la evolución de World Wide Web, HTML comenzó a proliferar muy rápidamente.

Pronto, las compañías comenzaron a crear exploradores de internet con el fin de poder visualizar documentos HTML tales como páginas web. A finales de 1995 [2], sorprendentemente el buscador de internet que dominaba el mercado era el conocido Netscape que actualmente está muy distanciado del popular Internet Explorer. Netscape fue el primer buscador capaz de soportar Javascript, imágenes animadas y marcos HTML. A partir de aquí comenzó una especie de batalla entre los diferentes exploradores que dominaban el mercado, lo que les condujo a crear sus propios elementos HTML que solo eran compatibles con su producto. Como consecuencia de esto, HTML comenzó a estar fragmentado y los creadores web empezaron a ver que sus páginas web no se visualizaban correctamente en todos los exploradores. Esto era un gran inconveniente ya que aumentó la dificultad y el tiempo de diseño en crear webs capaces de ser visualizadas correctamente en cualquier soporte. Hoy en día podemos ver que aún existen problemas de incompatibilidad de ciertos exploradores con ciertos lugares web continúa existiendo (30).

Frente a este problema, la compañía World Wide Web Consortium (W3C) trabajó en la correcta estandarización de HTML. Fueron publicadas varias versiones a finales de los noventa con el fin de ofrecer referencias oficiales a los autores de páginas web. La sucesión histórica de los estándares más relevantes comenzó con HTML 2.0 en Septiembre de 1995, HTML 3.2 en Enero de 1997 y HTML 4.01 en Diciembre de 1999. Actualmente aún se encuentra en desarrollo una versión definitiva de HTML 5.0 (30).

3.2.2.2. Ventajas de HTML

Como ya hemos comentado, HTML es el lenguaje de marcas en el que se fundamenta la World Wide Web. Sus principales ventajas:

- Facilidad de uso y comprensibilidad para los autores web.
- Principal formato base para la gran mayoría de navegadores web.
- Uso muy extendido a nivel mundial

A pesar de esto, HTML tiene el inconveniente de ser un lenguaje muy básico cuando se trata de aplicaciones complejas y también conlleva el problema de que su contenido no puede ser procesado como tal por programas. Por ello en la aplicación de nuestro

proyecto se utiliza el lenguaje Python para generar el servidor y HTML únicamente sintetiza el diseño grafico de la web para que pueda ser visualizada por un navegador.

3.2.2.3. Servidor web

A la hora de poder visualizar la interfaz gráfica de nuestro servidor web en el buscador de cualquier ordenador, ha sido necesario el uso de un lenguaje de marcas. A pesar de que el servidor web es utilizado en un entorno local, sigue los mismos estándares de un servidor colgado en la red por lo que se justifica el uso del lenguaje HTML para su programación.

4. DESCRIPCIÓN DE LA SOLUCIÓN HARDWARE IMPLEMENTADA

4.1. Introducción

Con el objetivo de dar una visión más clara al lector de cómo se estructura el proyecto, se ha dividido la descripción de componentes entre parte hardware y parte software. Ya que existe una estrecha relación entre ambas secciones, veremos que las figuras que esquematizan las interconexiones de los subsistemas tienen estructuras muy parecidas.

A diferencia de la parte software del proyecto, cuyo desarrollo parte prácticamente desde cero. En el hardware ya se partía de una plataforma de vuelo y una estación de tierra en la que ya era posible realizar pruebas de seguimiento del avión de forma manual. A continuación se hace una lista de los componentes hardware que ya existían y los añadidos durante el desarrollo del proyecto.

Tabla 3. Hadware existente e incorporado al proyecto

Hardware existente	Componentes añadidos
▪ Plataforma de vuelo: ○ *Aeronave y superficies de control* ○ *Sistema de radio control* ○ *Ordenador a bordo (PC/104)* ○ *Módulo de radio*	▪ Placa computadora Raspberry Pi
	▪ Base robotizada para la Antena de alta ganancia y servomotores
▪ Antena de alta ganancia para 2,4 GHz (sin base robotizada)	▪ Router wifi
▪ Módulo de radio para la Antena de alta ganancia	▪ Brújula digital

La labor por tanto en términos de hardware, consiste en ensamblar los nuevos componentes al sistema ya existente para conseguir pasar de una estación de tierra con antena de control manual a una estación de tierra autónoma con una antena independiente.

En los siguientes apartados se describirá el hardware ya existente directamente relacionado con el desarrollo de la antena autónoma, y por supuesto cada uno de los componentes añadidos.

4.2. Componentes de la estación de tierra autónoma

4.2.1. Sistema embebido

La parte hardware más importante en el desarrollo en nuestro proyecto es la placa computadora en donde se programará y procesará el código. Además de necesitar un micro controlador que lleve a cabo esta tarea, se requiere que posea puertos de comunicación a periféricos y que ocupe el menor espacio posible. Por ello se emplea un sistema embebido que pueda dar solución a las siguientes especificaciones:

- Procesador suficiente potente para ejecutar código con varios hilos de ejecución y un servidor web.
- Posibilidad de comunicar tres subsistemas por comunicación Serie.
- Salida Ethernet (RJ-45).
- Dimensiones reducidas para poder ser encajonado con facilidad.
- Capaz de soportar un sistema operativo y que posea salida gráfica que facilite la programación.
- Solución económica en la medida de lo posible.

Como ya se describió en el Capítulo 2.3, la solución adoptada como sistema empotrado es una Raspberry Pi modelo B que embebida dentro de una carcasa de diseño específico como el que se muestra en la siguiente figura, se ensamblará fija dentro de la caja solidaria a la antena.

Figura 18. Carcasa de diseño específico que aloja la Raspberry Pi

Aunque las especificaciones técnicas detalladas de la Raspberry Pi ya fueron expuestas en el Capítulo 2.3, a continuación las características más influyentes en nuestro proyecto:

Tabla 4. Características técnicas de la Raspberry Pi Modelo B para nuestra aplicación

Sistema embebido: Raspberry Pi Modelo B	
CPU	ARM1176JZF-S a 700 MHz (familia ARM11)
USB 2.0	2 (vía hub USB integrado)
Salida de video	HDMI (rev1.3 y 1.4)
Almacenamiento	SD / MMC / ranura para SDIO
Conexión a red	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel	8 x GPIO, UART
S.O.	Raspbian, Debian, Linux
Dimensiones	85.60mm × 53.98mm
Alimentación	5 V vía Micro USB o GPIO header
Precio	35 $US (Solo la placa)

4.2.2. Comunicación por radio con el UAV

Antena de alta ganancia para 2,4 GHz

Para recibir la señal del avión, el módulo de radio irá conectado a una antena con amplificador alimentado a la tensión de red diseñada para trabajar a 2,4 GHz. Se trata de una antena montada sobre una plataforma con dos grados de libertad rotativos, que a su vez están gobernados por dos servo motores. El movimiento de rotación e inclinación de la antena se encargan de orientar la antena para que se maximice la intensidad de las señales recibidas.

En la siguiente figura se muestran tanto la antena real utilizada como una imagen que representan los grados de movimiento rotativo de la antena:

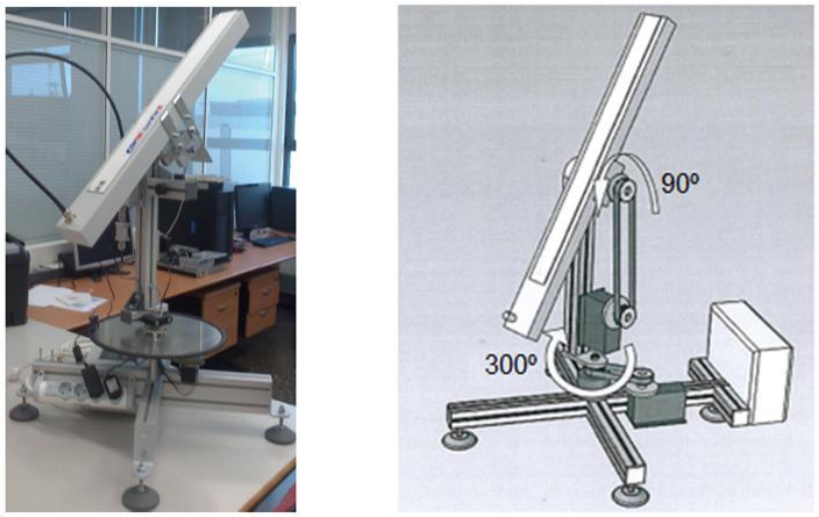

Figura 19. Antena de alta ganancia para 2,4 GHz

En la imagen de la derecha se observa que el giro en el eje perpendicular plano horizontal (o giro azimut) de la antena solo abarca 300 grados. Esto es debido a los topes mecánicos colocados para evitar romper los servos. Aún que esto podría suponer una limitación en el ángulo barrido por la antena, en la puesta en práctica no se trata de un gran inconveniente ya que la antena se colocara de tal forma que no se lleguen a alcanzar los grados de este ángulo muerto. Por otro lado, en el ángulo de inclinación solo se barren 90 grados ya que combinado con el azimut se consigue barrer el espacio necesario.

Módulo de radio conectado a la Antena de alta ganancia

A la hora de escoger un módulo de radio adecuado para la recepción y envío de señal entre el avión y la antena existen una serie de especificaciones que se deben cumplir:

- Posibilidad de trabajar en la banda de frecuencia 2,4 GHz.
- Poseer un alcance mínimo de entre 4-8 km en espacio abierto.
- Unas dimensiones y peso máximo determinado (14 x 7 cm y 100 g).
- Consumo máximo de potencia de 500mW.
- Posibilidad de elección de canales.
- Transferencia de datos mediante protocolo Puerto Serie (RS-232).

Solución adoptada:

Módulo de radio ***XStream TM OEM RF Module + MaxStream XIB-R Interface Board***

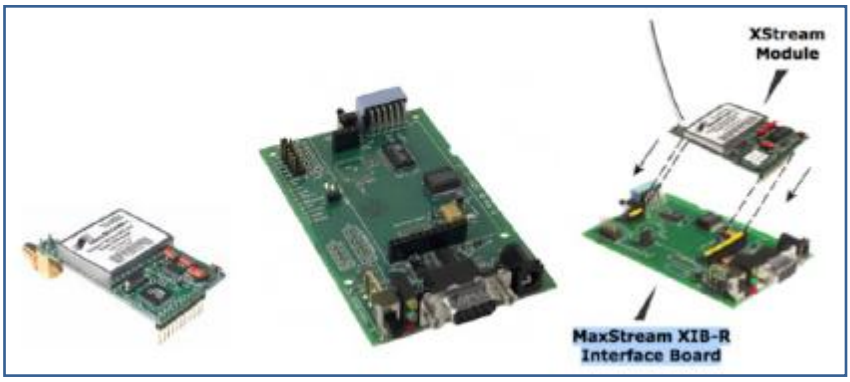

Figura 20. Módulo de radio XStream TM OEM + MaxStream XIB-R Interce Board

Las especificaciones técnicas del módulo de radio escogido cumplen con los requerimientos y se muestran en la Tabla 4.2:

Tabla 5. Características del módulo de radio utilizado

Módulo de radio: XStream TM OEM + MaxStream XIB-R Interce Board Systems		
Frecuencia		2.4 GHz
Alcance		16 km
Dimensiones	Ancho	10,16 cm
	Largo	6,4 cm
Consumo		50 mW
Nº de canales		175
Peso		47 g
Interfaz de datos		RS-232

4.2.3. Transmisión por wifi

Para el enrutamiento de los paquetes recibidos y transmitidos por el radio enlace es necesario tener conectado un Router wifi a nuestro sistema embebido que cumpla con los siguientes requisitos:

- Posibilidad de conexión Ethernet (RJ-45) con la placa computadora.
- Protocolo soportado UDP para el envío de paquetes.
- Protocolo DHCP para la asignación dinámica de IP a los clientes conectados.
- Dimensiones reducidas para su encajonado en la antena autónoma.
- Antena externa para mejorar la transmisión desde fuera del cajón.
- Solución de bajo coste.

Solución adoptada: *Router* ***Edimax BR-6228nS***

Figura 21. Router Edimax BR-6228nS

La solución adoptada cumple con los requisitos necesarios como se observa en las características más relevantes se detallan en la siguiente tabla:

Tabla 6. Características del enrutador utilizado

Enrutador: Edimax BR-6228nS		
Interfaz hardware	Puerto WAN	1x RJ-45
	Puertos LAN Fast Ethernet	4x RJ-45
Conexiones LAN		Hasta 253 clientes
Estándares compatibles		802.11b/g/n
Protocolo de red soportado		IP, TCP, UDP, DHCP, entre otros.
Antena		1x 3dBi fija
Flujo máximo de datos		150Mbps
Dimensiones		134 x 110 x 26 mm
Alimentación		DC 5V, 1A
Precio		16€ con IVA

4.2.4. Control del movimiento de la antena

Brújula digital

Con el objetivo de tener una fuente de referencia global de orientación para nuestra antena autónoma, se plantea la idea de implementar una brújula digital que vaya conectada al sistema embebido y sujeto solidariamente a la estructura de la base robótica de la antena. De este modo el sistema podrá tener en todo momento una lectura de la orientación de referencia respecto de un norte.

A la hora de escoger un dispositivo u otro, había ciertas especificaciones que se deberían cumplir en cuanto a los requerimientos de nuestro sistema:

- Posibilidad de comunicación Serie.
- Pequeñas dimensiones.
- Bajos niveles de consumo ya que será alimentada desde la propia Raspberry Pi.
- Capacidad de re calibración.
- Bajo coste.

Solución adoptada: ***Brújula digital CMPS10***

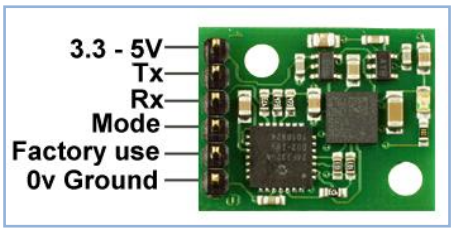

Figura 22. Brújula digital CMPS10 en Modo

En la imagen se muestran los pines de la brújula para su funcionamiento en modo Serie que será mediante el cual la conectaremos a la Raspberry Pi en sus pines GPIO.

Las especificaciones técnicas de la brújula escogida se muestran en la siguiente tabla:

Tabla 7. Características de la brújula digital utilizada

Brújula digital: Electronic Compass CMPS10		
Resolución		0.1°
Precisión	Horizontal	1%
	Inclinada +/- 60°	1.5%
Salida	Modo Puerto Serie	9600 baud
		Sin bit de paridad
		2 bit de stop
Dimensiones		24 x 18 mm
Alimentación		3.3-5V, 25mA Typ
Precio		15,10€ sin IVA

Servo motores

Para el control de los dos grados de libertad rotativos de la plataforma base de la antena, son necesarios dos motores que sean capaces de leer señales digitales y transformarlas en movimientos de rotación a la posición requerida. Para ello se utilizarán como alternativa dos servomotores cuyas características requeridas se describen a continuación.

- Torque suficiente para rotar la antena robotizada (T>20kg·cm aprox.)
- Tamaño y peso reducido para facilitar su ensamblaje.
- Capaz de operar a cierta temperatura debido al auto calentamiento (Tª>40ºC).
- Modulación digital por ancho de pulso (PWM).
- Coste reducido.

Solución adoptada: ***Hitec HSR-5980SG - Steel Gear Robot Servo***

Figura 23. Servomotor Hitec HSR-5980SG

Las características técnicas de los servos seleccionados que cumplen con las especificaciones son las siguientes:

Tabla 8. Características de la brújula digital utilizada

Servomotor: Hitec HSR-5980SG		
Sistema de control	por Ancho de Pulso	1500 µs
Rango de Tª de trabajo		-20°C a 60°C
Torque desarrollado	Alimentado a 6 V	24 Kg·cm
	Alimentado a 7.4 V	30 Kg·cm
Dimensiones		40 x 20 x 37 mm
Peso		70 g
Alimentación		4.8 V a 7.4 V, > 5.2 A
Precio		82€/unidad con IVA

Para enviar las señales digitales con la posición que se quiere a los servomotores se utiliza una tarjeta de control conectada a la Raspberry por comunicación Serie y ajustada de la siguiente manera:

- Baudios: 115200
- Bits por dato: 8
- Bit de stop: 1
- Paridad: Ninguna
- Control de flujo: Ninguno

4.2.5. Componentes del UAV relevantes en el desarrollo

A pesar de que nuestro proyecto se centra en el desarrollo de la estación de tierra autónoma sin influir en la plataforma de vuelo, a la hora de trabajar y realizar pruebas en laboratorio se han utilizado instrumentos como la unidad inercial (IMU) y un dispositivo GPS conectado a esta, que forman parte del UAV y que merece la pena describir.

GPS

El sistema global de posicionamiento (Global Positioning System) es un sistema que permite determinar en todo globo terráqueo la posición de un objeto con precisiones de unos metros, e incluso, en ciertas aplicaciones, de centímetros.

El GPS funciona mediante una red de 24 satélites en órbita sobre el planeta tierra, a 20.200 km, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor que se utiliza para ello localiza automáticamente como mínimo tres satélites de la red, de los que recibe unas señales indicando la identificación y la hora del reloj de cuada uno de ellos. A partir de la posición de cada satélite y de la posición relativa del receptor GPS con respecto de ellos se obtienen las posiciones absolutas reales del punto de medición (31).

Principio físico de funcionamiento

La situación de los satélites puede ser determinada de antemano por el receptor con la información del llamado almanaque (conjunto de valores con 5 elementos orbitales), parámetros que son transmitidos por los propios satélites. La colección de los almanaques de toda la constelación se completa cada 12-20 minutos y se guarda en el receptor GPS.

La información que es útil al receptor GPS para determinar su posición se llama efemérides. En este caso cada satélite emite sus propias efemérides, en la que se incluye la salud del satélite (si debe o no ser considerado para la toma de la posición), su posición en el espacio, su hora atómica, información doppler, etc.

El receptor GPS utiliza la información enviada por los satélites (hora en la que emitieron las señales, localización de los mismo) y trata de sincronizar su reloj interno con el reloj atómico que poseen los satélites. La sincronización es un proceso de prueba y error que en un receptor portátil ocurre una vez cada segundo. Una vez sincronizado el reloj, puede determinar su distancia hasta los satélites, y una esa información para calcular su posición en la tierra.

Cada satélite indica que el receptor se encuentra en un punto en la superficie de la esfera, con centro en el propio satélite y de radio la distancia total hasta el receptor. Obteniendo información de dos satélites se nos indica que el receptor se encuentra sobre la circunferencia que resulta cuando se intersecan las dos esferas.

Si adquirimos la misma información de un tercer satélite notamos que la nueva esfera sólo corta la circunferencia anterior en dos puntos. Uno de ellos se puede descartar

porque ofrece una posición incongruente (fuera del globo terráqueo, sobre los satélites). De este modo ya tendríamos la posición en 3D. Sin embargo, dado que el reloj que incorporan los receptores GPS no está sincronizado con los relojes atómicos de los satélites GPS, los dos puntos determinados no son precisos.

Teniendo información de un cuarto satélite, eliminamos el inconveniente de la falta de sincronización entre los relojes de los receptores GPS y los relojes de los propios satélites. AL no estar sincronizados los relojes entre ellos, la intersección de las cuatro esferas con centro en estos satélites es un pequeño volumen en vez de ser un punto. La corrección consiste en ajustar la hora del receptor de tal forma que este volumen se transforme en un punto (31). Y es en este momento cuando el receptor GPS puede determinar una posición 3D exacta (latitud, longitud y altitud).

En el supuesto de que el receptor consiguiera obtener señal de 4 satélites la determinación de la posición del receptor es equivalente a la resolución de un sistema de cuatro ecuaciones con cuatro incógnitas como el siguiente:

$$d_1 = \sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2} + c\Delta t = c\left(t_{r,1} - t_1\right)$$

$$d_2 = \sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2} + c\Delta t = c\left(t_{r,2} - t_2\right)$$

$$d_3 = \sqrt{(X_3 - x)^2 + (Y_3 - y)^2 + (Z_3 - z)^2} + c\Delta t = c\left(t_{r,3} - t_3\right)$$

$$d_4 = \sqrt{(X_4 - x)^2 + (Y_4 - y)^2 + (Z_4 - z)^2} + c\Delta t = c\left(t_{r,4} - t_4\right)$$

Donde:

- X_i, Y_i, Z_i con i=[1…4] es la posición de cada uno de los satélites.
- "c" es la velocidad de la luz ($3 \cdot 10^8$m/s)
- t_i (en la base de tiempos del satélite) es el instante de tiempo en el que el satélite "i" manda la señal.
- $t_{r,i}$ es el instante de tiempo en el que el receptor recibe la señal del satélite "i".
- Δt es el offset que posee el receptor respecto del sistema de tiempos del satélite.
- d_i es la pseudodistancia al satélite "i" (distancia real + el offset debido a la sincronización de relojes) (32).

Fuentes de error

Las posibles fuentes de error que pueden interferir en el funcionamiento del GPS son (31):

- *Ionosfera* (Retraso de la señal)
- *Efemérides* (Envío de información no actualizada)
- *Reloj del satélite* (Inexactitudes causadas por ruidos y derivas en los relojes atómicos de los satélites)
- *Distorsión multibanda* (Reflexión de la señal en tierra, edificios, etc. Que provoca que la señal llegue más tarde)
- *Troposfera* (Retraso de la señal)
- *Errores numéricos*

Unidad inercial

Una IMU o unidad inercial de medida (Inertial Measurement Unit) es en realidad una composición de sensores que incorpora 3 acelerómetros y 3 giróscopos. A continuación pasamos a describir cada uno de ellos para, posteriormente, explicar cuál es la función que desempeñan en conjunto.

- Acelerómetro: Miden la aceleración a través de la deformación que sufre el soporte de una masa inercial, empleando sensores del tipo galga extensométrica, que han sido serigrafiados y calibrados durante el proceso de fabricación del sensor (33). Son unidireccionales y, en presencia del campo gravitatorio, dan como medida la intensidad de éste.

- Giróscopo: Sensor de velocidad angular, capaz de medir velocidades de rotación alrededor de un eje. Existen diversos tipos de giróscopos, cada uno de los cuales está regido por principios físicos diferentes. Los más usados son: mecánicos, ópticos y electrónicos. Los giróscopos mecánicos son los más antiguos mientras que los ópticos y los electrónicos son más modernos (34).

Como puede verse, cada uno de estos sensores tiene una propiedad generalmente unidireccional. De esta forma, la razón de unir tres de cada tipo en un mismo encapsulado responde a la necesidad de hacer una composición vectorial que mida

una magnitud espacial. Efectivamente, la disposición de estos seis sensores será ortogonal tres a tres, siguiendo las direcciones de los ejes principales de inercia de la aeronave.

Este sensor es empleado como sistema de navegación inercial. Las informaciones de cada transductor son utilizadas por un ordenador que filtra las medidas y las combina convenientemente para computar posición, velocidad, orientación (actitud) y el índice de giro de un vehículo. Sin embargo, estas medidas poseen imperfecciones tales como ruidos y desviaciones, que conducen a errores en el estado de navegación crecientes con el tiempo (32).

Actualmente este tipo de sensor incorpora ya en su interior el computador (micro controlador), dejando disponible al usuario tanto las medidas de los transductores independientes en crudo, como las aproximaciones a actitud, velocidad, etc., de las que hablábamos en el párrafo anterior. Con el fin de compensar las deficiencias de las IMU, se suelen emplear otros sensores independientes. Estos sensores son normalmente llamados *asistentes a la navegación* y se caracterizan por su estabilidad a largo plazo, compensando precisamente los errores a corto plazo aportados por los sensores inerciales (32).

Solución adoptada: **IG-500N de SBG Systems**

Figura 24. Unidad Inercial IMU IG-500N de SBG

Las características más importantes de la IMU utilizada para aplicaciones aéreas, como lo es la medida de datos del UAV para la telemetría, se resumen en la siguiente tabla:

Tabla 9. Características de IG-500N de SBG

Unidad inercial: IG-500N de SBG Systems		
Precisión de posición	SPS	2.5 CEP
Frecuencia de actualización máxima de la posición	On board	100 Hz
	Externo	500 Hz
Precisión estática	Pitch/Roll	<0.5 deg
	Yaw	<1 deg
Precisión dinámica de Actitud y Rumbo		1 deg RMS
Resolución angular de Actitud y Rumbo		0.05 deg
Rangos dinámicos	Pitch	± 90 deg
	Roll/Yaw	± 180 deg
Frecuencia de actualización máxima de Actitud y Rumbo	On board	100 Hz
	Externo	500 Hz
Interfaz digital		Seria, Can y USB
Voltaje operativo		3.3 a 30V
Consumo		800mV
Salidas proporcionadas:	Ángulos de Euler	
	Quaternion	
	Matrices de rotación	
	Velocidad 3D	
	Posición 3D	
	Datos de sensores calibrados	
	Datos de sensores en crudo	
	Datos del GPS	
Precio		3760 € sin IVA

4.3. Interconexiones entre componentes

Una vez han quedado todos los componentes de expuestos, queda interconectarlos entre ellos mediantes los protocolos de comunicación pertinente para conseguir el resultado requerido.

El sistema final con el que se realizarán las pruebas de vuelo se puede dividir en tres módulos bien diferenciados.

- UAV: Constituye la plataforma de vuelo.
- Punto de Monitorización: Representa el otro extremo en el proceso de comunicación.
- Antena Autónoma: Comprende el ensamblaje de los componentes descritos en la primera parte de este capítulo.

En la siguiente figura se trata de esquematizar de forma gráfica como se intercomunican todos los subsistemas del proyecto.

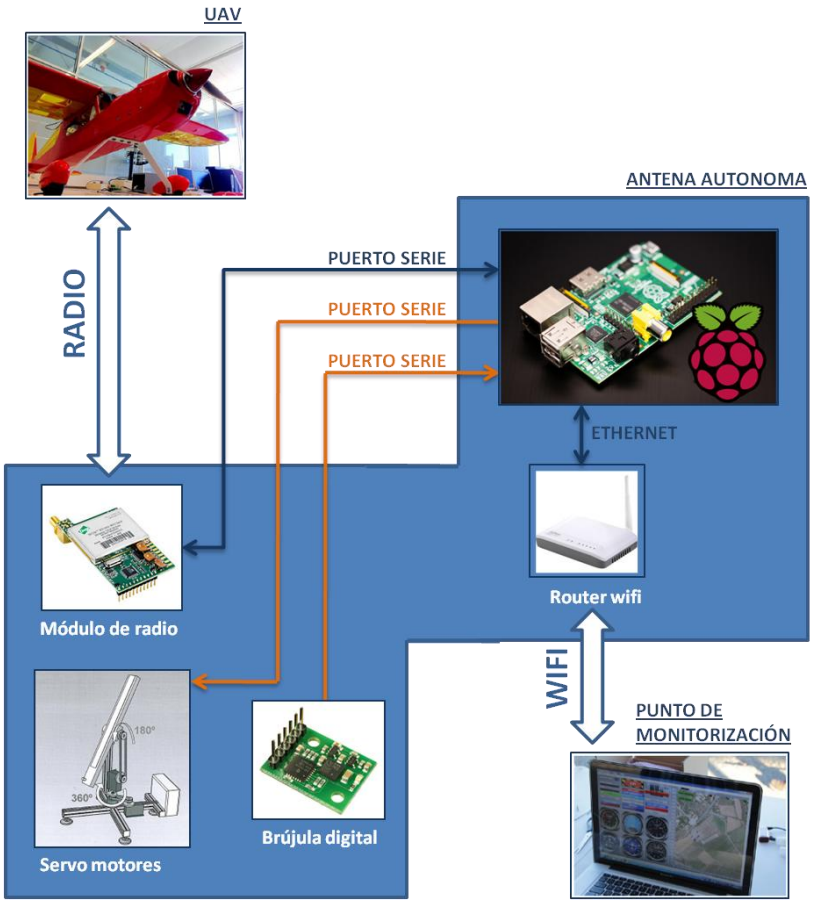

Figura 25. Esquema de interconexiones entre subsistemas Hardware

En primer lugar, la comunicación entre el UAV y la Antena autónoma se establece por radio en la banda de frecuencia 2.4 GHz por medio de los módulos de radio *XStream TM OEM* instalados tanto en el avión como en la antena de alta ganancia.

Por otro lado la comunicación del bloque antena con el resto de puntos de monitorización que quieran conectarse a nuestro sistema se realiza vía wifi. En particular, por protocolo de transferencia de paquetes UDP.

Y adentrándonos un poco más en detalle en las interconexiones de nuestro sistema, tenemos la placa computadora Raspberry Pi conectada al resto de elementos:

- Router *Edimax BR-6228nS*: Conexión con cable de Ethernet (RJ-45) que es reconocido automáticamente por la Raspberry Pi. Dentro de la configuración del Router, se asigna una IP fija local a la MAC de la Raspberry Pi de modo que siempre se puede acceder a ella a través de la misma dirección dentro de nuestra red local. Estos es útil especialmente para tener una dirección fija del host del Servidor Web.
- Radio Enlace *XStream TM OEM RF Module*: Conexión serie con cable USB/RS232 reconocido automáticamente por la Raspberry Pi tanto por "path" (directorio) como por "id". La transferencia se establece a 19200 baudios, sin bits de paridad ni de parada, se leen 8 bits por dato y sin control de flujo.
- Servos *Hitec HSR-5980SG*: Al igual que con el radio enlace, se establece una conexión serie con cable USB/RS232 con el controlados de los servos también reconocido automáticamente por la Raspberry Pi. El envío de las señales de control se establece a 115200 baudios, sin bits de paridad, con 1 bit de parada, se leen 8 bits por dato y sin control de flujo.
- Brújula digital *CMPS10*: Conexión serie entre los pines de GPIO de la Raspberry Pi correspondientes a la UART y los pines de la brújula digital Tx y Rx en el Modo Serie. En este caso el envío de las señales de control se establece a 9600 baudios, sin bits de paridad, con 2 bit de parada, se leen 8 bits por dato y sin control de flujo. Hay que señalar que la brújula debe ser alimentada por lo que también se conecta el pin del GPIO que corresponde a 5V.

A continuación se muestra un plano de la Raspberry Pi modelo B utilizado en donde se señala claramente cada una de las salidas utilizadas para la comunicación con los dispositivos explicados.

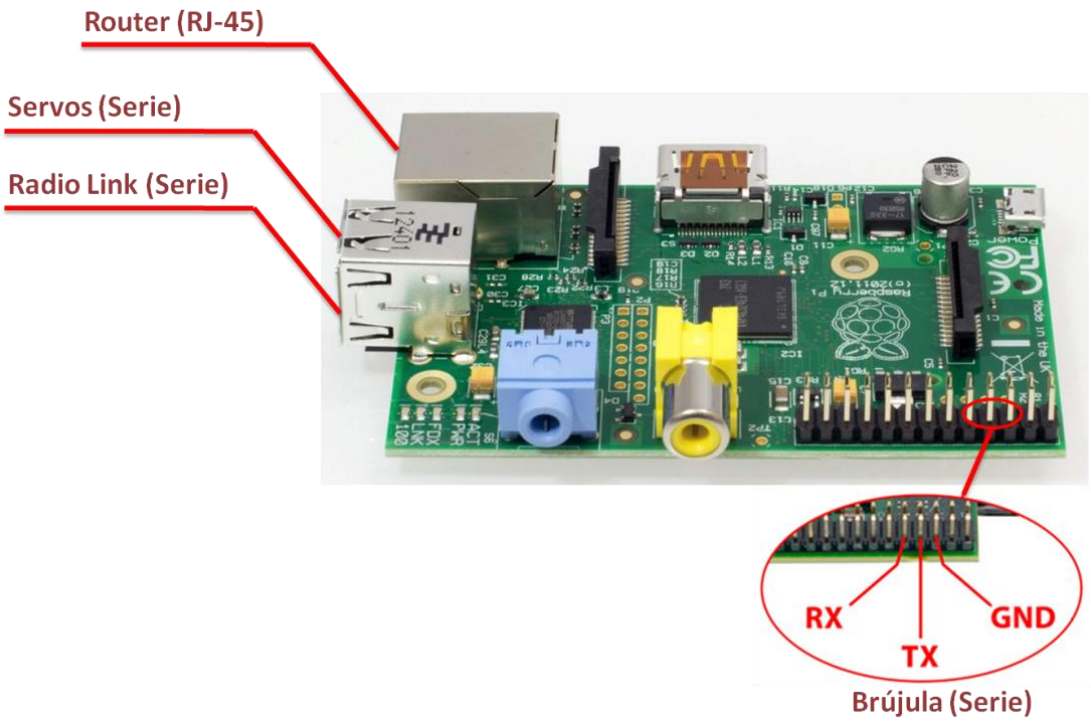

Figura 26. Salidas para las comunicaciones de la Raspberry Pi

5. DESCRIPCIÓN DE LA SOLUCIÓN SOFTWARE IMPLEMENTADA

5.1. Introducción a la solución general adoptada

A continuación se describe la solución que se adopta en términos de software para implementar las funciones que realizan tanto la gestión de la telemetría como el control de la antena. Este trabajo realizado puede considerarse un proyecto prácticamente de desarrollo software, ya que se partía de una base hardware tanto en la plataforma de vuelo como en la estación de tierra ya montadas y ensambladas. Por ello, se puede considerar este capítulo expone la descripción de la parte más importante en cuanto a carga de trabajo durante la elaboración experimental del proyecto.

Además de este capítulo, se ha escrito un Manual de Usuario y un Manual de Programación donde se detalla la programación del código desde el principio. Por ello aquí únicamente nos limitaremos a hablar de la estructura básica e inclusión del programa dentro del sistema embebido, los hilos de ejecución implementados y una breve descripción de la librería de funciones elaborada.

5.1.1. Disposición del software por aplicaciones

El desarrollo del proyecto se implementa de manera que todos los componentes quedan como un solo bloque como se muestra en la Figura 27.

Una placa computadora Raspberry Pi constituye nuestro sistema embebido ya que es núcleo para el desarrollo software gracias a las ventajas que aporta expuestas en el Capítulo 2.3. Sincronizados por esta, se encuentran el resto de los subsistemas que gobiernan el correcto funcionamiento de la antena: Los servo motores, la brújula digital, el enlace de radio y el Router wifi).

Como se representa en el diagrama, el sistema operativo instalado en nuestra Raspberry Pi es una versión de Linux (Raspbian) diseñada específicamente para esta placa. Gracias que la GNU de Linux viene directamente con un compilador de lenguaje C, únicamente era necesario instalar un IDE (Entorno de Desarrollo Integrado) para escribir el código. Por motivos de eficiencia y en cuanto a la potencia del procesador de la Raspberry Pi, como editor de texto para el desarrollo del código hemos utilizado

el IDE Geany. Es una herramienta diseñada con las propiedades básicas de un entorno de desarrollo integrado que proporciona un IDE simple pero rápido, necesario para trabajar correctamente dentro de las capacidades de la Raspberry.

El lenguaje mayoritariamente utilizado es C. Con este se ha realizado desde cero la programación de todo el código referente a la gestión de la Telemetría y el Control de la Antena. Sin embargo, para la implementación del Servidor Web que se visualiza en el punto de monitorización, partimos de una aplicación ya creada en Python. Mediante el aprendizaje de conocimientos básicos sobre este lenguaje, se han modificado diferentes partes del código de manera que el servidor cumpliese con los requisitos que planteamos en los objetivos. Por este motivo, en lo relativo a la programación del Servidor Web detallaremos únicamente las modificaciones aportadas y nos limitaremos a explicar de forma breve la estructura general del programa.

En siguiente esquema se trata de representar de forma gráfica los componentes que acabamos de describir.

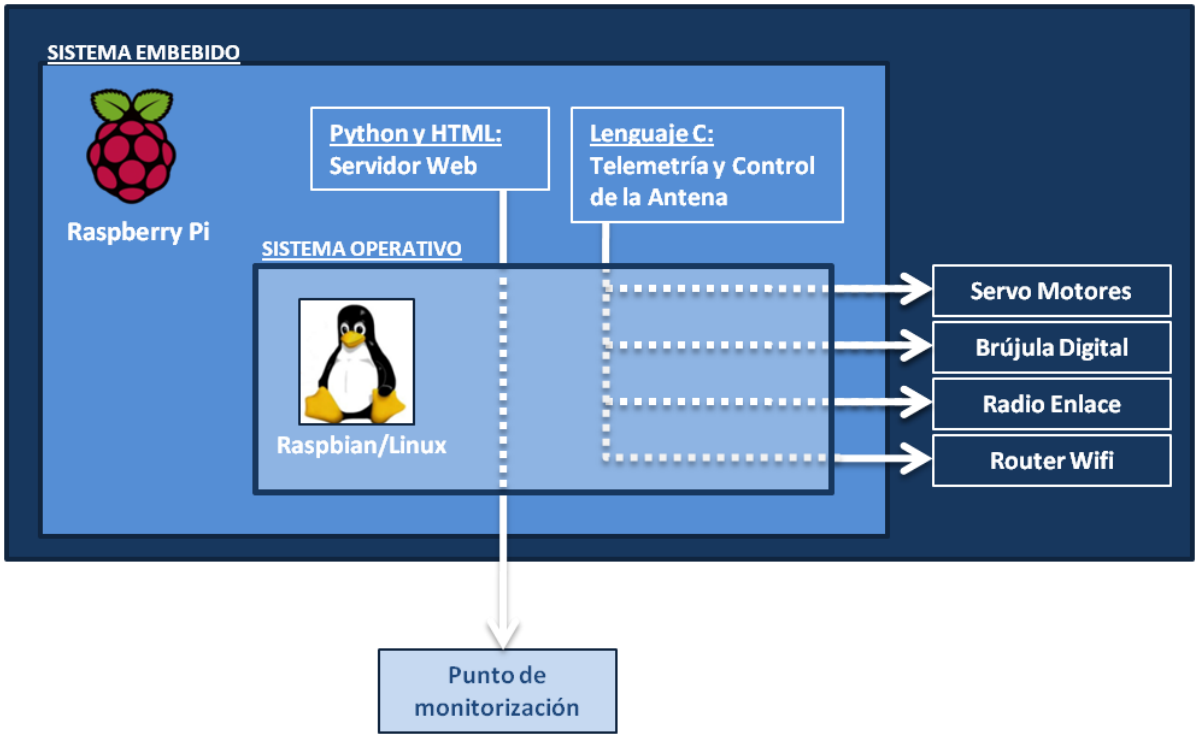

Figura 27. Esquema del sistema general Software

5.1.2. Estructura de los ficheros creados

El código en C de la gestión de la Telemetría y el Control de la Antena se estructura principalmente en dos módulos (módulo principal y librería elaborada) con sus respectivas cabeceras:

1. Un fichero principal “main.c” en donde se encuentra la función principal y cinco hilos de ejecución independientes gobernados por esta.
2. Un fichero cabecera “main.h” en donde simplemente se inicializan las cinco funciones de cada hilo de ejecución y el identificador necesario para la estrategia de no-interbloqueo MUTEX.
3. El fichero “library_functions.c” que contiene la librería elaborada con las funciones que utilizan los hilos.
4. Por último el fichero cabecera “library_functions.h” en donde además de inicializar todas las funciones de nuestra librería, también se declaran todas las variables globales y las definiciones (“defines”) utilizadas en el programa.
5. Además de estos cuatro ficheros, se incluye una carpeta “utilities” que no ha sido creada específicamente para este proyecto pero de la cual se utilizan algunas funciones que se explican con más detalle en el Manual de Programación.

5.2. Función principal e hilos de ejecución

Todo programa debe contener una función principal desde la cual se gestionan los recursos y el resto de funciones. En el caso de nuestra aplicación existe una función "main" que se encargará de gestionar los argumentos de entrada introducidos por el usuario en cada ejecución, pero sobre todo el papel de esta función principal es inicializar y lanzar los hilos de ejecución que trabajaran de forma independiente.

Para nuestra solución hemos hecho uso de cinco hilos, los cuales se iniciaran nada más ejecutar la función principal. Debido a su programación interna, si cada hilo es inicializado y ejecutado correctamente, no dejará de funcionar hasta que se detenga de forma externa al programa la propia función principal.

Los cinco hilos independientes que gobiernan el funcionamiento del programa son:

1. Gestión de la comunicación Puerto Serie de la Telemetría.
2. Gestión de la comunicación por UDP de la Telemetría.
3. Gestión del Control de la Antena.
4. Lectura continua de la Brújula digital.
5. Función de Rastreo en caso de pérdida de la señal del avión.

Los cuales están declarados de la siguiente manera:

```
void *Telemetry_SerialPortManagement();

void *Telemetry_UDPManagement();

void *Antenna();

void *UAV_scanning();

void *Compass_reading();
```

Figura 28. Diagrama de interconexiones entre las funciones del programa

En la Figura 4.2 se muestra un fragmento del código C de la cabecera "main.h" que corresponde a la inicialización de estas cinco funciones correspondientes a cada uno de los hilos. Como se puede ver, se trata de funciones sin argumentos de salida ni parámetros de entrada ya que la información que se extrae de ellas y que se utiliza

para la intercomunicación entre hilos, se almacena en variables globales como se describe a continuación.

5.2.1. Funcionalidad de cada hilo independiente

Una vez expuestos los módulos que trabajara de forma independiente durante la ejecución del programa, se va a explicar un poco más en detalle y de forma gráfica cuál es la principal función de cada uno de los hilos.

1. Gestión de la comunicación Puerto Serie de la Telemetría

Para la implementación de la telemetría que nos permite recibir y enviar datos del avión, intervienen dos de los hilos. En el primero se gestiona la comunicación por protocolo puerto serie (como se muestra en la Figura 29) entre la Raspberry Pi, que en nuestro caso se encarga de gestionar todos los dispositivos, y el módulo de Radio Enlace que se comunicará directamente con el UAV emitiendo y recibiendo a una frecuencia de 2,45GHz.

Figura 29. Esquema de la función Gestión Puerto Serie en la Telemetría

Esta función se encargará de abrir el socket necesario para la comunicación, establecer la velocidad de lectura en baudios, y configurar las líneas de lectura /escritura según convenga (por ejemplo haciendo estas líneas no bloqueantes en caso de que no pudiesen leer o escribir nada).

2. Gestión de la comunicación por UDP de la Telemetría

La otra función que interviene en la gestión de la telemetría corresponde a la siguiente parte de la transmisión que comunica de nuevo la Raspberry Pi a través de un Router Wifi con los Puntos de monitorización que se conecten a nuestra red local Wifi (Figura 4.4). Hay que destacar que en el proceso de repetir la información proveniente del

UAV se realiza mediante protocolo UDP (User Datagram Protol). Mediante este sistema de envío no se necesita tener una conexión previa con el receptor al contrario del protocolo TCP/IP. De este modo, nuestro sistema estará enviando continuamente paquetes con la información sin producirse errores de comunicación en caso de que se pierda algún paquete por el camino.

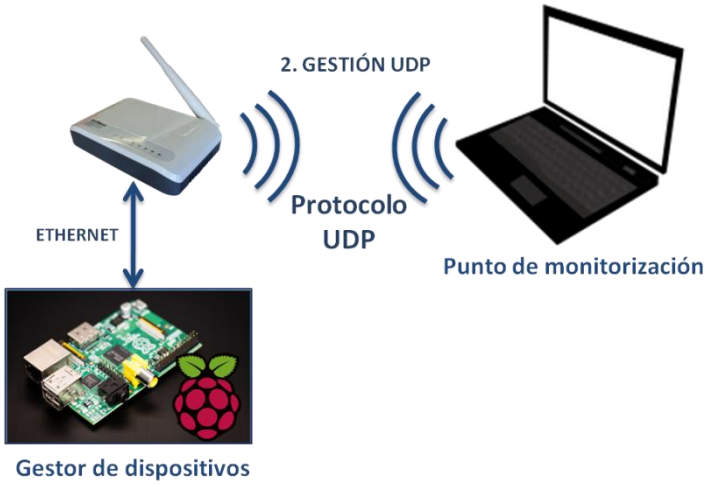

Figura 30. Esquema de la función Gestión UDP en la Telemetría

Para otras muchas aplicaciones puede ser de utilidad usar un protocolo de comunicación TCP/IP que verifique continuamente el correcto envío y recepción de información, ya que en caso de pérdida de un paquete este se reenvía hasta que se verifica la recepción. En nuestro caso se trata de una aplicación en donde la información sobre la localización del avión debe ser actualizada al instante. La prioridad es enviar en tiempo real esta información y no verificar su correcta recepción, ya que si se perdiese un paquete en medio de la transmisión simplemente influiría en la perdida de un instante de muestreo y sería solucionado con la recepción del siguiente paquete un instante después.

3. Gestión del Control de la Antena

Una vez gestionada la telemetría, el siguiente paso es utilizar correctamente los datos recibidos de la plataforma de vuelo para conseguir que la antena con dos grados de libertad rotativos apunte continuamente al avión. Entre las tramas recibidas se encuentran los datos de latitud, longitud y altura del UAV con respecto a la superficie de la tierra. Mediante una serie de aproximaciones y cambios de sistema de referencia

se obtienen el ángulo de azimut que gobierna el movimiento rotativo sobre el eje vertical de la antena, y el ángulo de inclinación que varía con la altura a la que se encuentra el UAV. Este método se explica con más detalle en el Manual de Programación.

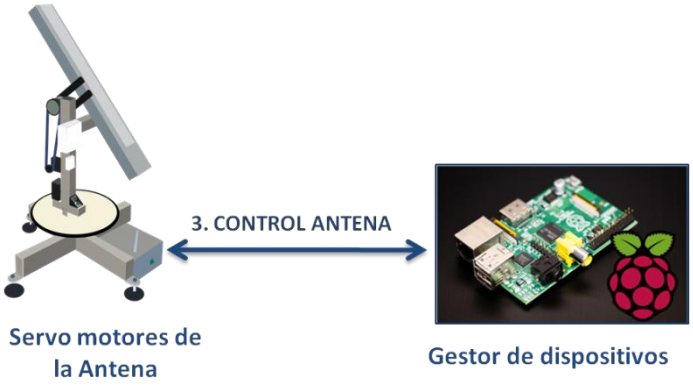

Figura 31. Esquema de la función Gestión Control de la Antena

En esta figura se muestra una vez más que es nuestra Raspberry Pi la que se encarga de gestionar el resto de los componentes. En este caso la comunicación se realiza mediante comunicación puerto serie al igual que con el módulo de radio de la telemetría. La Raspberry está conectada al controlador de los servos a los cuales envía el ángulo de posicionamiento necesario a cada instante.

4. Lectura continua de la Brújula digital

Para hacer el sistema más robusto se utiliza una brújula digital solidaria a la antena que entrega en todo momento el ángulo global de esta. A la hora de localizar el avión es necesario referenciar la posición de la antena. Por ello a la hora de inicializar el proceso de control de los movimientos rotativos de la antena es necesario leer el ángulo que nos da la brújula en el instante inicial lo cual gestiona la orientación que tiene la antena de forma autónoma.

Por ello era necesario lanzar un hilo independiente que este continuamente leyendo y guardando los valores de la brújula en una variable global de modo que ese valor esté disponible en tiempo real para el resto de funciones que lo necesiten.

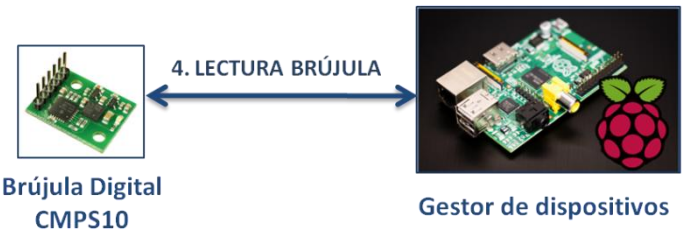

Figura 32. Esquema de la función Lectura de la Brújula digital

En la Figura 32 se muestra el modelo de brújula digital CMPS10 utilizado y comunicado por protocolo puerto serie a nuestro gestor de dispositivos Raspberry Pi.

5. Función de Rastreo en caso de pérdida de la señal del avión

Por último, con motivo de mejorar una vez más la robustez del sistema, se implementa una función de rastreo que se activa de forma automática en el momento que se pierde la señal del avión. Se trata de un hilo que se arranca desde la inicialización del programa, pero que se encuentra a la espera de que se dejen de recibir datos validos de localización por la telemetría para comenzar a funcionar. De este modo, en el momento que el avión no deja de ser localizado, la antena empieza a realizar un barrido a velocidad intermedia y no para hasta recibir de nuevo un dato valido lo que implica que el UAV ha sido encontrado de nuevo.

5.2.2. Descripción de la sincronización de hilos

Una vez explicada la función básica de cada hilo, es interesante ver como se sincronizan, cada cuanto se ejecutan y como se comunican entre ellos mediante el uso de buffers y variables globales.

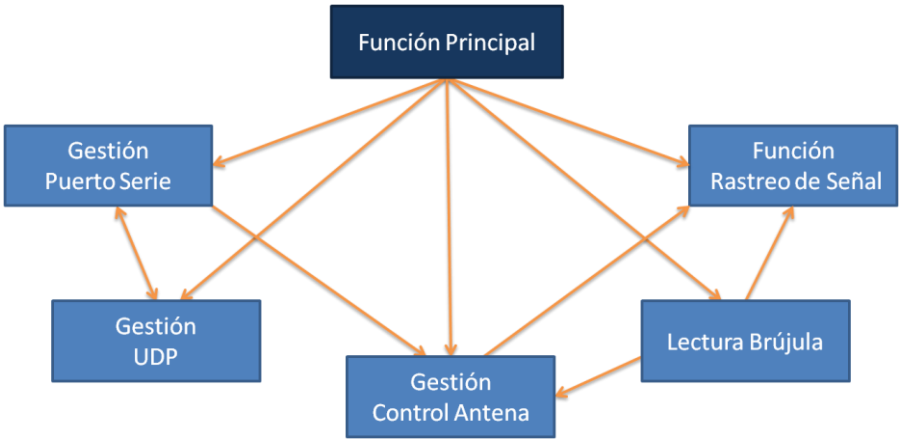

Figura 33. Diagrama de interconexión entre la función principal e hilos

Con motivo de explicar un poco más en detalle esta sincronización entre funciones, se ha realizado un diagrama donde se muestra la función principal, los cinco hilos y las interconexiones entre ellos, ya sea debido a la llamada de una dentro de otra o por el hecho de intercambiar información mediante el uso de variables globales. Según el sentido de las flechas que conectan los diferentes bloques se puede identificar en qué sentido va el flujo de datos o cuál es la función que realiza la llamada.

- Función principal: Como ya se ha comentado al principio de este capítulo, la función principal se encarga de efectuar la llamada a los cinco hilos nada más arrancar el programa.
- Gestión Puerto Serie: La gestión de puerto serie de la telemetría se comunica con la función que gestiona la transmisión UDP mediante el uso de dos buffers. Esta función se ejecuta cada 25 milisegundos para mejorar el muestreo en la lectura del puerto serie por lo que es necesario el uso de un buffer a la hora de depositar información y de que otra función independiente la recoja. Además este hilo también se comunica con la función de gestión del control de la antena ya que va guardando los parámetros de localización del avión en una variable de tipo estructura para su posterior lectura y procesado.
- Gestión UDP: Esta función únicamente se comunica con la gestión del puerto serie de la telemetría tanto para recibir la información relacionada con la localización del UAV y de la propia antena, como para enviar posibles acciones de control en el otro sentido. En este caso, esta función tiene un ciclo de ejecución cada 100 milisegundos por lo que irá leyendo/escribiendo con una frecuencia más lenta la información en los buffers.
- Gestión Control Antena: Para el control de la antena necesitamos recibir información tanto sobre la localización del avión suministrada por la función gestión de puerto serie de la telemetría, como de la lectura de la brújula digital con la orientación de la antena para conocer su norte de referencia. Está función se ejecuta cada 100 milisegundos y pone en funcionamiento la función de rastreo cuando se dejan de recibir datos válidos sobre la localización del avión.

- Lectura Brújula: Está función que se ejecuta cada 100 milisegundos indefinidamente desde el arranque del programa, se limita a almacenar de forma continua el valor del ángulo de la orientación de la antena respecto del norte global. De modo que cuando otras funciones requieran de este parámetro, simplemente tienen que leer la variable global donde se almacena.

- Función Rastreo de señal: Y por último la función encargada de gestionar la búsqueda del UAV en cada de pérdida de su localización. Esta función a diferencia del resto, se encuentra permanentemente a la espera de que una variable de tipo "flag" (booleana) indique la pérdida del avión para comenzar a trabajar. El rastreo comienza un tiempo "t" después de no recibir datos sobre el avión. Este tiempo es configurable por el usuario a través del servidor web en el arranque del programa y es necesario que sea al menos mayor de 2 segundos para el correcto funcionamiento de la antena.

5.2.3. Gestión de los tiempos de ejecución de cada hilo

En ingeniería de control es importante controlar los tiempos de ejecución de los diferentes módulos que constituyen el sistema.

En este proyecto, la sincronización entre los hilos independientes y el intercambio de información en tiempo real requiere que hilo cumpla rigurosamente con los tiempos de ciclo anteriormente expuestos. Para llevar a cabo esta tarde es necesario hacer uso de una serie de funciones relacionadas con el muestreo de tiempos de programa.

La estructura básica es tomar el tiempo de reloj en la primera línea dentro del bucle de la función y posteriormente inmediatamente después de la última línea. Con esta diferencia de tiempos y con la función *usleep()* de C es posible crear una espera justo al final de cada ciclo antes de empezar el bucle de nuevo. Ya que nuestra aplicación requiere tiempos de ciclo del orden de milisegundos, es suficiente trabajar con la función *usleep()* con la precisión de un microsegundo.

5.2.4. Solución para el interbloqueo de los hilos: Estrategia MUTEX

Para la sincronización entre hilos de ejecución, además de controlar los tiempos de ciclo es necesario evitar los posibles interbloqueos que puedan producirse entre las

funciones de cada hilo. Esto puede ocurrir en el caso de que dos funciones estuviesen accediendo a la misma variable global a la vez y se condicionase la ejecución de una función con la otra.

Por ello, es imprescindible utilizar una estrategia de exclusión MUTEX que gracias a un funcionamiento similar a los semáforos en programación, evita que dos o más funciones que trabajen en paralelo modifiquen la misma variable global a la vez.

El ejemplo más representativo del uso de esta estrategia en nuestro programa ocurre durante la comunicación entre la función de gestión del puerto serie de la telemetría y la función de gestión de UDP. Debido a la diferencia entre tiempos de ciclo, es necesario el uso de dos buffers para la sincronización del envío de la información. El buffer "buf_RadioLink" donde se reciben las tramas del UAV, se refresca cada 25 milisegundos. Sin embargo, el buffer de lectura del gestor UDP "buf_sendUDP" tiene un refresco cada 100 milisegundos. En la Figura 4.8 se representa gráficamente el proceso descrito.

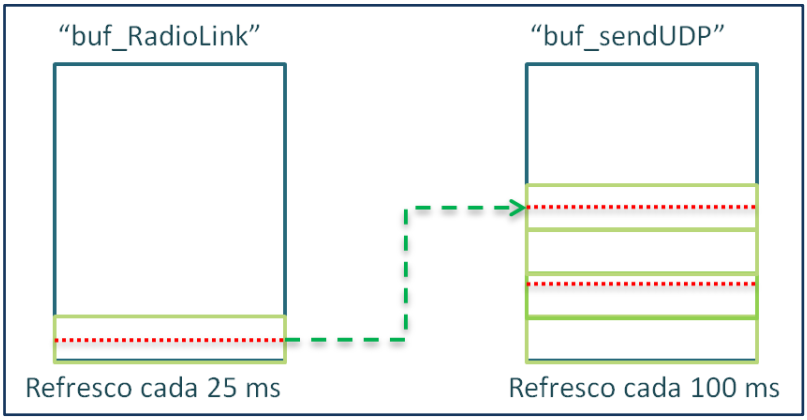

Figura 34. Transferencia de información entre buffers en la gestión de la telemetría

A la hora de apilar la información en el "buf_sendUDP" iremos modificando un puntero "pointer_bufUDP" definido como variable global. Es aquí donde necesitamos hacer uso del algoritmo de exclusión mutua MUTEX. Consiste en bloquear el posible acceso a la variable común durante el tiempo que está siendo modificada.

5.3. Librería desarrollada

Además de utilizar librerías estándar de C a la hora de tratar con cadena de caracteres o implementar funciones matemáticas, ha sido necesario elaborar una biblioteca con funciones propias específicas para el funcionamiento del programa.

En el Manual de Programación anexado se encuentra el desarrollo detallado sobre el funcionamiento de cada una de las 26 funciones que constituyen esta librería. Por ello en este apartado realizaremos una breve enumeración y descripción cualitativa de la librería describiendo a grandes rasgos la tarea de las funciones desde el punto de vista funcional.

La librería consta de:

- Una función que gestiona los parámetros de entrada de la función principal facilitados por el usuario por medio del servidor web.
- Un total de cinco funciones encargadas de abrir y configurar de forma específica los sockets necesarios para la comunicación, tanto vía Puerto Serie como vía UDP, del sistema embebido con los distintos dispositivos conectados.
- Cuatro funciones para gestionar la lectura y la escritura por puerto serie y UDP en la telemetría.
- Dos funciones relacionadas con la detección de errores. Una para verificar la validez de paquetes en la recepción mediante el cálculo del CRC, y otra función que reporta el tipo de error producido durante la ejecución del programa en caso de fallo.
- Una función que empaqueta la información sobre la posición de la antena y su orientación para que posteriormente pueda ser enviada por UDP.
- Dos funciones relativas a la lectura de la brújula digital.
- Cinco funciones necesarias para transformar los parámetros latitud, longitud y altura del UAV en ángulos de giro azimut y elevación de los servos.
- Una función de encargada del escaneo de señal para el hilo de Rastreo.
- Y por último cinco funciones que se utilizan para el control de los movimientos rotativos perpendiculares al eje vertical a partir del cálculo del ángulo azimut.

5.4. Servidor Web

Nuestro sistema embebido se encuentra programado y encajonado junto con el resto de componentes necesarios dentro de la estructura de la antena. Por este motivo, cuando el sistema está montado para realizar una prueba de vuelo, la unidad de control Raspberry Pi no es accesible por medios físicos.

Para poder poner en funcionamiento el programa que gestiona la telemetría y el control de la antena es necesario, por tanto, la implementación de un servidor web que permita controlar las acciones básicas de encendido/apagado desde un punto de monitorización externo. De este modo, si alguien se conecta vía wifi a la red local creada por nuestro enrutador podría visualizar la interfaz gráfica que se muestra en la Figura 35. Interfaz gráfica del usuario con la antena mediante el servidor web para introducir los parámetros necesarios y poner en funcionamiento el programa.

5.4.1. Justificación del la solución de partida

Como se ha mencionado con anterioridad, la idea de implementar un servidor web para mejorar la interfaz con el usuario parte de una aplicación ya creado en Python. Este servidor web previo ya contenía las funcionalidades necesarias para la introducción de parámetros y acciones de control. A partir de él se han implementado algunas mejoras que son de utilidad para mejorar la robustez del sistema.

La principal justificación del uso de Python es que, aparte de las ventajas que ofrece este lenguaje de alto nivel ya comentadas en el Capítulo Otros lenguajes utilizados, es capaz de crear y poner en marcha un servidor web simple con unas pocas líneas de código. Esto es posible debido a la potente librería que proporciona Python. Por tanto, gracias a la programación orientada a objetos y a la posibilidad de crear clases con herencia a partir de otras, es interesante crear un servidor web personalizado según los requerimientos de nuestra aplicación.

5.4.2. Apariencia gráfica y descripción de los parámetros

En la Figura 35 se muestran todos los parámetros de entrada al programa y los botones que permiten al usuario controlar el funcionamiento básico de la antena.

Figura 35. Interfaz gráfica del usuario con la antena mediante el servidor web

La funcionalidad de cada botón del servidor web es la siguiente:

- Update: Actualiza una base de datos ligada al servidor que alberga los parámetros introducidos por el usuario y permite que estos queden guardados cada vez que se cargue el servidor web.

- Telemetry: Es el encargado de arrancar la aplicación que gestiona la telemetría y el control de la antena. Antes de ponerlo en funcionamiento deben existir los

parámetros de entrada en la base de datos para poder, en caso contrario el código lanzará un error por insuficiencia de argumentos de entrada.

- Stop: Una vez arrancada la aplicación existe la posibilidad de detenerla pulsado este botón de stop. Como método de seguridad para evitar que el programa pueda ser ejecutado más de una vez en la placa computadora, lo que implicaría varios programas a la vez enviando ordenes a la antena. Se ha implementado un "flag" al programa en Python que impida que la aplicación pueda ser ejecutada más de una vez.

- Reboot: Nos permite reiniciar la Raspberry Pi si fuese necesario.

- Shut Down: Este botón apaga la Raspberry Pi directamente.

Y por otro lado la función de cada una de los parámetros de entrada es:

- Latitude, Longitude y Altitude: Estos tres parámetros de entrada corresponden a la localización de la antena en el momento de realizar la prueba de vuelo. Previo a la puesta en marcha del programa, es imprescindible introducir la localización exacta de la antena ya que de no ser así, el programa calcularía un vector posición incoherente del avión con respecto de la antena. Como ya se ha explicado en relación al botón de "update". De no ser modificados los parámetros de entrada, estos se refrescarán con los parámetros que ya existían en la base de datos. Esta acción es interesante para los siguientes parámetros de velocidad y tiempo de comienzo de rastreo, pero no para los datos de localización de la antena ya que cambiarán en función de donde se sitúe la antena durante una prueba de vuelo.

- Scan speed: Corresponde a la velocidad de giro que pretendemos que tenga la antena a la hora de realizar una operación de rastreo de la señal en caso de pérdida de la localización del UAV. Es valor estará acotado entre un máximo y un mínimo para evitar fallos mecánicos del sistema. Por norma general, la velocidad de rastreo suele fijarse en la mitad de la velocidad que tiene la antena en situación normal de posicionamiento.

- Scan time: Además de la velocidad de giro para la función de rastreo, otro parámetro interesante es el tiempo que deseamos que la antena espere después de haber perdido la localización del avión para empezar a rastrear. Para tiempos iguales o menores que un segundo, la antena estaría continuamente dando por perdida y encontrada al avión e impediría un correcto funcionamiento de la antena. Por ello, es recomendable un tiempo aproximado de cuatro segundos para evitar que con el mínimo intervalo de pérdida de paquetes la antena lance la función de rastreo.

- IP Address: Por último, uno de los parámetros de entrada más importantes correspondientes con la gestión de la telemetría del avión es la dirección IP a la que se desean repetir por UDP los datos recibido por el radio enlace. Para mejorar la robustez del sistema se implementa la posibilidad de repetir la telemetría a más de un punto de monitorización a la vez como se esquematiza en la Figura 36. Esquema de la comunicación por UDP con varias direcciones IPFigura 36. El funcionamiento es el siguiente:
  - Si se introduce una o más direcciones válidas con las directrices de nuestra red local (192.168.2.XX), la telemetría se repetirá a dichas direcciones.
  - En caso de no introducir ninguna dirección válida, la telemetría será automáticamente gestionada para ser enviada por *broadcast* a todo el rango de direcciones de nuestra red local (192.168.2.255). De este modo, se asegura que en caso de introducir direcciones no válidas, los paquetes sigan repitiéndose.

Figura 36. Esquema de la comunicación por UDP con varias direcciones IP

5.4.3. Estructura y descripción de los ficheros implementados

En cuanto al código en Python del Servidor Web, consta de cuatro ficheros importantes que gestionan el servidor:

- El fichero principal “mainweb.py” con el que arranca el servidor que con escasas cinco líneas de código se encarga de importar las clases que van a ser utilizadas e iniciar el servidor web en el puerto especificado.
- El segundo archivo “webserver.py” es realmente el esqueleto del programa que soporta el servidor web ya que gestiona las llamadas a los eventos que van siendo lanzados desde el servidor, administra la base de datos que almacena los parámetros de localización del avión entre otros, y lanza la interfaz en HTML para su visualización desde cualquier navegador web.
- El fichero “eventos.py” se encarga de gestionar de forma independiente las distintas peticiones al servidor tales como identificar si se quiere actualizar la base de datos y acceder a esta, o cuando se solicita ejecutar una acción de control start/stop del programa. También contiene la línea que se encarga de suministrarle los argumentos de entrada en el arranque de la función “main.c” gracias a la función en Python “os.system()”.

- La gestión de la lectura y escritura en la base de datos se lleva a cabo gracias al fichero “sqlite.py” el cuál es llamado desde “eventos.py” en cada acceso los datos.

Aparte de usar Python, también se utilizan ficheros scripts “index.html” en lenguaje de marcas HTML para la visualización gráfica dentro de la carpeta /webs. Por último, también se incluye una pequeña base de datos “DataAntena.db” dentro de la carpeta /Database editable con SQLite que permite almacenar las variables que el usuario introduce en el arranque de la aplicación.

6. PRUEBAS Y MEJORAS REALIZADAS

Después de conocer los principales objetivos a alcanzar en el proyecto, y las herramientas software y hardware a utilizar. En este capítulo se describe el proceso de desarrollo de la parte experimental desde que se empieza a programar el código hasta la última prueba realizada con existo que da por alcanzados los objetivos propuestos.

6.1. Primer programa

En primer lugar se da una etapa de programación de un código que cumpla con los requisitos básicos de funcionamiento, para poder realizar las primeras pruebas de comunicación entre el avión y la antena. En esta etapa se van desarrollando de forma independiente la gestión de la comunicación entre módulos para la telemetría, y el algoritmo de gestión del control de la antena.

La parte limitante a la hora de realizar la primera prueba era la gestión de la telemetría ya que involucraba el poder recibir tramas del avión y extraer su localización GPS para permitir que la antena pudiese trabajar con esos datos. En un principio, la estructura de esta función que gestionaba la telemetría era completamente secuencial tal y como se representa en la siguiente figura.

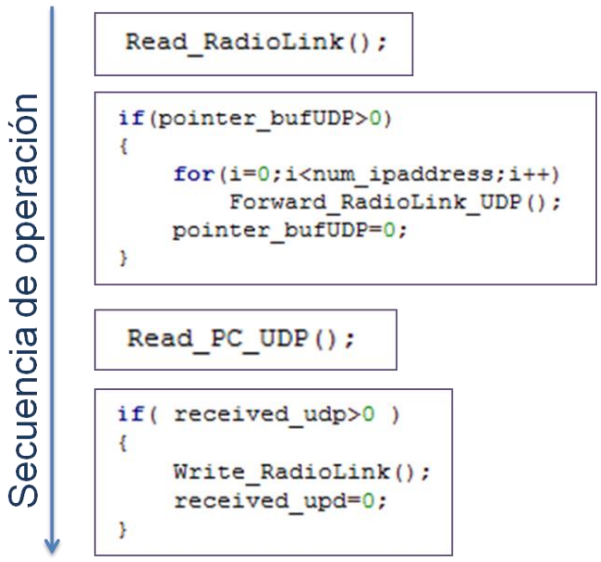

Figura 37. Esquema del primer código secuencial de la gestión de la telemetría

El funcionamiento de este proceso constaba únicamente de un solo hilo ejecutándose ininterrumpidamente del siguiente modo:

1) Lectura del radio enlace por puerto serie.
2) En caso de que se lea algo, repetir esta información por UDP.
3) A continuación leer si se reciben paquetes por UDP.
4) En caso afirmativo, reenviar esta información por el radio enlace.

Una vez esta parte del código funcionaba, había que extraer los datos de localización del avión de las tramas recibida por el radio enlace y compartirlas en una variable global con el otro hilo de ejecución que gestiona el control de la antena.

Tras ensamblar ambos hilos de ejecución llamados por una función principal *main()*, ya era posible comenzar a testear la efectividad del programa e ir mejorar su robustez.

6.2. Primer test

Tras tener operativa una primera versión del programa se procedió a realizar una primera prueba al aire libre con el fin de testear la comunicación entre el ordenador a bordo del avión y la antena.

- Desarrollo

La prueba consistió en colocar la antena en un entorno abierto aunque con obstáculos físicos tales como arboles y edificios de pequeño tamaño, y simular el recorrido del avión únicamente desplazando el ordenador a bordo (PC/104) conectado a la unidad inercial IMU y al GPS para dar la posición. Independiente a la antena había un punto de monitorización al que se le repetía la información por wifi.

- Logros

A pesar de que esta primera prueba fue poco exitosa debido a los fallos de comunicación entre antena y "avión", se comprobó que cuando la transmisión funcionaba, la antena era capaz de localizar su situación y apuntarlo. Eso confirmaba que el algoritmo fundamental de detección de tramas y control de la antena funcionaban correctamente.

- Mejoras a realizar

Ya fuese debido a la comunicación entre los módulos de radio del UAV y la antena o entre la antena y el punto de monitorización por UDP, se perdían muchos paquetes de información y se plantearon una serie de tareas para mejorar el sistema y hacerlo más robusto de cara al siguiente test:

  - Comprobar si la perdida de tramas era por UDP o por el Radio Link.
  - Implementar una secuencia de rastreo en caso de pérdida de señal.
  - Dividir tareas en hilos de ejecución independientes y cambiar tiempos de ejecución de los mismos.
  - Implementar el envío de la posición y orientación por UDP para su visualización en el punto de monitorización.

6.3. Segundo test

Una vez llevada a cabo la implementación de las mejoras propuestas después del primer test, se propone realizar un segundo test del mismo modo pero cambiando a un escenario de mayor superficie y sin obstáculos como es un campo de futbol.

- Mejoras implementadas
  - ✓ Para esta segunda prueba se concluye que las pérdidas importantes de tramas son debidas al Radio Link al comprobar que se siguen perdiendo el mismo número de tramas sustituyendo el envío wifi por una conexión Ethernet (RJ-45) entre el Router y el punto de monitorización. Para mejorar en cualquier caso la robustez de nuestra comunicación por UDP, es aquí donde se implementa la posibilidad de conectar más de un cliente a nuestro servidor-antena mediante la introducción previa de la dirección IP del cliente en el Servidor Web.

  - ✓ Se implementa y se prueba la secuencia de rastreo de modo que cuando la antena pierde la señal del avión durante un ensayo, automáticamente comienza a rastrear el entorno hasta volver a recibir una señal válida. Con esta nueva función se mejora en gran medida la autonomía de la antena.

- ✓ Se cambia la estructura del código, de modo que en vez de existir únicamente los dos hilos anteriores, se divide el programa en cinco hilos de ejecución y se controlan sus tiempo de ejecución:

    1) Gestión de la comunicación serie con el Radio Link.
    2) Gestión de la comunicación UDP.
    3) Gestión del control de la antena.
    4) Función de rastreo.
    5) Lectura constante de la brújula digital.

- ✓ Se implementa el envió de la posición y orientación de la antena por UDP, de modo que paralelamente a enviar la localización del avión, se repiten paquetes también con esta información. Gracias a esta mejora conseguimos visualizar además del avión, la orientación que tiene la antena en tiempo real. Lo que nos verifica que la antena sigue al avión una vez más como se había proyectado.

- Mejoras a realizar

Aún con las nuevas mejoras implementadas, el sistema seguía perdiendo un considerable número de paquetes durante el ensayo. La principal tarea a continuación sería revisar la comunicación que existía entre el Radio Link y el módulo de radio del avión para conseguir solventar este problema. Además de esta tarea, se propusieron adicionalmente las siguientes:

- Comprobar la comunicación entre el UAV y el Radio Link de la antena.
- Implementar un “flag” en el Servidor Web que impidiese iniciar dos veces el programa.
- Comprobar la linealidad de los ángulos leídos en la brújula digital.
- Comprobar canales del Radio Link y del Router ya que ambos operan a 2,4GHz y pueden interferirse entre ellos.
- Añadir parámetros de entrada para la función rastreo desde el Servidor Web.

6.4. Tercer test

Tras revisar en detalle la línea de comunicación entre el UAV y el Radio Link de la antena, comprobando todas las conexiones desde el ordenador a bordo PC/104 hasta la conexión de la antena con el radio enlace. Se observo que el cable coaxial que unía la antena externa con el módulo de radio dentro del PC/104, estaba dañado. Al sustituir este cable y probar nuevamente la comunicación observamos que prácticamente no se perdían paquetes y que este había sido potencialmente una de las fuentes de error en la comunicación.

Por tanto esta última prueba se llevo a cabo una vez se habían realizado las siguientes modificaciones:

- Mejoras implementadas
  - ✓ Se sustituyó el cable coaxial dañado ya que contribuía a un gran número de paquetes perdidos en la comunicación, y que aún con el control manual de la antena, no corregiría ese problema de no ser reparándolo.
  - ✓ Quedo implementado el "flag" de inicio de programa que evitaba que la aplicación fuese puesta en marcha más de una vez desde el Web Server lo que provocaría errores en el control de la antena con dos o más programas ejecutando a la vez.
  - ✓ También se realizo un test detallado con la brújula digital para comprobar de su linealidad. Primero se colocó la brújula en un soporte no metálico suficientemente separado de cualquier parte ferro magnética de la estructura de la antena, y se compararon los valores de lectura de la brújula con una referencia de ángulos reales en todo el rango de operación con intervalos de 10°. Haciendo una tabla con todos los valores se observó que los valores leídos eran prácticamente lineales salvo tolerancias de >4° en los puntos más extremos lo cual es completamente aceptable en el contexto de nuestra aplicación.
  - ✓ El objetivo de comprobar los canales de operación del Radio Link y el Router es escogerlos los más separados posible para que aún

trabajando a la misma frecuencia de 2,4 GHz, intentar que intentar que interfirieran lo mínimo posible uno sobre el otro. Ya que el Radio Link era menos accesible a la hora de configurar sus canales, se fue probando con los diferentes canales del Router wifi hasta encontrar uno que funcionase de forma óptima.

- ✓ La quinta y última mejora implementada después del segundo test fue añadir una serie de parámetros al Servidor Web, que el usuario pudiese introducir por pantalla, tales como la velocidad de rotación en la operación de rastreo, y el tiempo que debía esperar la antena antes de comenzar esta búsqueda de señal.

Una vez estuvieron listas todas estas tareas propuestas se desplazó la antena nuevamente a una superficie abierta tal como una pista de atletismo, y se procedió a realizar el mismo ensayo que en los dos test anteriores.

Desde el principio se observó una notable mejora en la recepción de tramas directamente al punto de monitorización y conforme el ordenador a bordo se alejaba a gran distancia de la antena, se veía como está en ningún momento perdía señal y seguía apuntando de forma permanente al “avión”.

Con este tercer test se dieron por finalizadas las pruebas de operatividad de la antena autónoma para la gestión de la telemetría por cumplir con los objetivos proyectados inicialmente.

7. MEJORAS EN FUTUROS PROYECTOS

7.1. Mejoras en el software

Acciones de control

En la función que gestiona la telemetría están implementados dos sentidos de comunicación. Por un lado la transferencia de los paquetes provenientes del avión recibidos en el radio enlace y que son repetidos por protocolo UDP gracias al Router wifi conectado a nuestros sistema. Y por otro lado la lectura por UDP de llegada de posibles paquetes para reenviarlo por el radio enlace de nuevo hacia el avión.

A pesar de tener un protocolo de comunicación bidireccional implementado, a día de hoy únicamente se han realizado pruebas recibiendo información del avión. En futuros desarrollos del proyecto se podría hacer uso de ambos sentidos de la comunicación enviando acciones de control al UAV desde la estación de tierra, o incluso mejorando la capacidad de este otro sentido de transferencia quizás podría llegar a ser posible cargar un software de control predictivo que permitiese realizar ensayos de vuelo sin necesidad de control remoto.

Algoritmo de localización universal

Nuestro algoritmo actual para obtener los ángulos azimut e inclinación de la antena de alta ganancia consisten en una aproximación de regresión lineal en la cual se contempla únicamente una franja del mapa terrestre y sus alrededores. En el caso de que quisiésemos trasladarnos con la antena a otro lugar suficientemente alejado (otra ciudad o incluso otro país), habría que modificar el software para cambiar las coordenadas en las que se desea realizar el ensayo de vuelo.

Una posible mejora en el software es utilizar esas coordenadas como una variable que pudiese ser introducida como parámetro de entrada para el usuario a través del servidor web. De esta manera cada vez que cambiásemos de localización habría que actualizar las coordenadas de dos puntos entre los cuales se quiere realizar la aproximación lineal.

Otra posible mejora y que involucraría un cambio importante en el código sería modificar completamente el algoritmo existente e implementar uno nuevo capaz de generar estos dos ángulos a partir de las posiciones globales de localización del avión (latitud, longitud y altura) de una forma universal. Uno de los principales problemas que habría que abordar a la hora de implementar un método universal es el hecho de que la tierra no tiene un radio constante. Como se observa en la siguiente figura explicativa, el radio R_{T1} correspondiente a la posición en donde se encuentra la proyección del UAV sobre la superficie de la tierra no sería igual al radio R_{T2} correspondiente a la situación de la antena. Por este motivo poseyendo las coordenadas globales del UAV (latitud, longitud y altura), no sería suficiente para determinar de manera exacta los vectores que determinarían el ángulo azimut y el de inclinación. Sería necesario introducir en una base de datos un mapeado de la superficie terrestre con sus respectivos radios de modo que fuesen accesibles por el algoritmo.

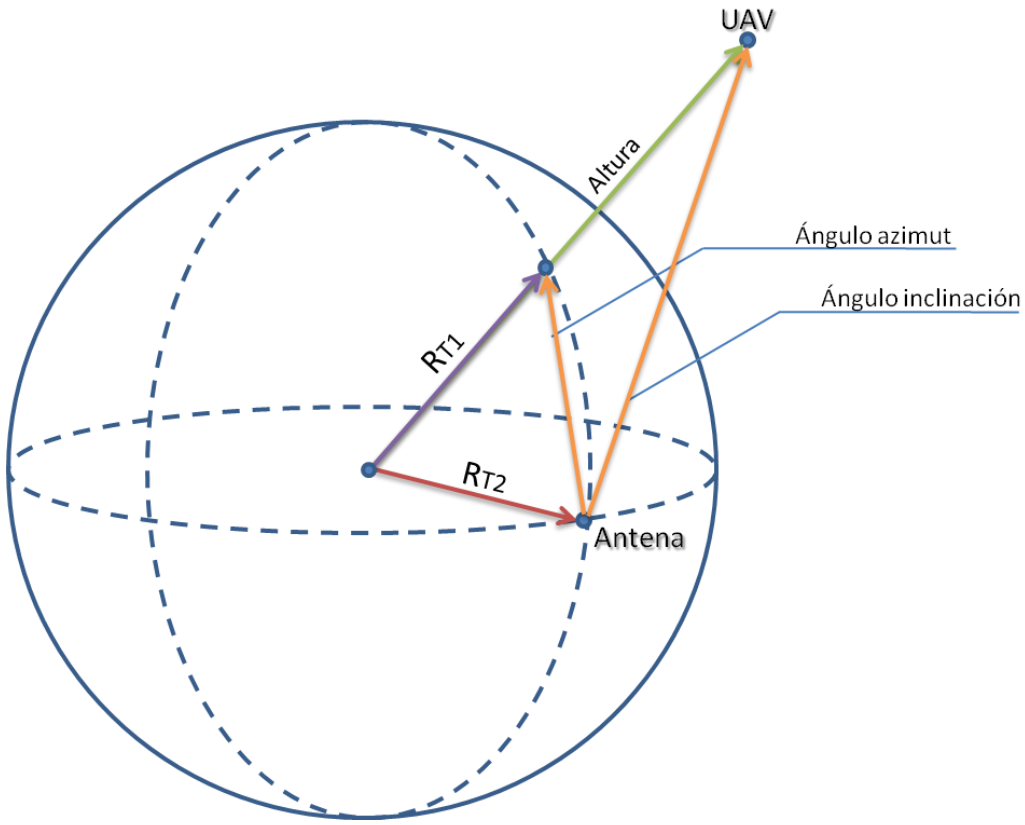

Figura 38. Representación esférica de la localización del UAV y de la antena en la tierra

Sin embargo, en una hipótesis de aproximar el radio de la tierra a su radio medio conocido (6371 km), la varianza que posee el radio terrestre está en el rango de $R_T \in [6353,6384]$ km lo que supone una varianza máxima de 13 metros. Aunque no se ha profundizado en el estudio de esta posible solución, se prevé que cometiendo este error se siga consiguiendo un algoritmo capaz de hacer que la antena apunte correctamente al avión. Como se ha dicho, esto simplemente es una hipótesis para futuras mejoras en la robustez del programa y aún está por probar.

7.2. Mejoras en el hardware

Otra frecuencia de operación

Nuestra frecuencia de transmisión entre el UAV y nuestra antena de alta ganancia de la estación de tierra es 2.4 GHz, la misma banda de frecuencia a la que opera el wifi, por lo que en entornos con redes de este tipo cercanas podrían emitirse interferencias y provocar fallos en la transferencia de paquetes.

Una posible mejora en cuanto al hardware actual de nuestro sistema sería trabajar en otro rango de frecuencia que también fuese libre como puede ser 5.1 GHz. Para ello habría que modificar las dimensiones de la antena de alta ganancia de forma que permitiese radiar en una nueva banda de frecuencia.

También hay que tener en cuenta la limitación que tienen las frecuencias libres, ya que normalmente están limitadas entorno a 100mW de potencia y en el caso de querer utilizar nuestro UAV en aplicaciones de transferencia de video en tiempo real sería necesario trabajar a otras frecuencias que permitiesen sobrepasar una potencia de 1500mW.

Modificar los motores

Actualmente, en nuestra base robotizada de la antena de alta ganancia hay instalados dos servomotores descritos en el capítulo 4. En ocasiones, estos servomotores se recalientan en exceso debido a largos periodos de uso lo que provoca que dejen de funcionar hasta su posterior enfriamiento.

Una posible solución a este problema sería o bien usar servo motores de mayor potencia que sean capaces de vencer mayores resistencias en el par entregado, e

incluso utilizar motores paso a paso que además reducirían la limitación en el rango de ángulo de giro del servomotor.

Unidad GPS en la antena

En nuestro sistema actual, la localización de la antena es necesaria para poder calcular la posición del avión correctamente respecto de la antena. Para conseguir esta localización, actualmente está implementado en nuestro programa que el usuario introduzca mediante el web server la latitud, longitud y altura a la que se encuentra la antena.

Una de las más importantes mejoras propuestas sería la de añadir una unidad GPS solidaria a la antena de alta ganancia y modificar el software de manera que en vez de obtener los parámetros al iniciar el programa por el servidor web; la latitud, longitud y altura fueran leídas directamente de su GPS.

Con esta solución abordaríamos dos puntos importantes. Por un lado se evitaría la tarea de tener que localizar por un elemento externo (p. ej. Google Earth) la posición aproximada de la antena en un punto de la superficie. Esto implica un margen de error que luego repercute en el correcto seguimiento del avión. Y por otro lado se conseguiría un sistema más independiente y más robusto ya que mejoraría la autonomía de la antena lo cual es el principal objetivo en el planteamiento de este proyecto.

8. CONCLUSIONES

Tras haber expuesto los principales objetivos de este proyecto en el capítulo 1.2 de la memoria y haber expuesto las etapas de su desarrollo, a continuación se valora la adecuación de las soluciones tomadas y el cumplimiento de estos objetivos.

- Implementar en una antena de alta ganancia un sistema autónomo de seguimiento de un vehículo aéreo no tripulado (UAV).

La idea de implementar una estación de tierra autónoma pasa por conseguir que la antena de alta ganancia que recibe los paquetes de información con la telemetría del avión, sea capaz de estar orientada continuamente hacia este de manera que se consiga la mayor potencia de transmisión en dicha comunicación.

Con el software implementado y los componentes hardware que forman nuestro sistema, tras haber realizado varias pruebas en superficies abiertas que simulaban ensayos de vuelo, se ha conseguido verificar que la antena era capaz de localizar y apuntar de forma constante al UAV.

Está función también se ha visto reforzada por la implementación de un algoritmo de rastreo que es capaz de gestionar barridos automáticos de búsqueda ante la pérdida de señal del avión, y que en las pruebas se comprobó su gran efectividad tanto al arrancar el programa sin conocer la posición previa del UAV, como en pleno vuelo ante una pérdida de la señal.

- Independizar la antena de conexiones físicas con los puntos de monitorización que reciben y analizan la telemetría del UAV.

No menos importante era la idea de poder situar la antena de alta ganancia en un lugar adecuado para mejorar su visibilidad con el avión y al mismo tiempo no tener que trasladar los equipos de monitorización hasta el punto donde se situaba la antena, o que estuvieran enclavados por conexiones físicas como funcionaba previamente con cable Ethernet.

Gracias a la gestión de los paquetes de información recibidos por el radio enlace y su posterior repetición vía Router mediante protocolo UDP, conseguimos que cualquier

ordenador con conexión wifi pueda conectarse a la red local de nuestro sistema y controlar la puesta en marcha y las funciones de la antena.

Durante las pruebas realizadas se comprobó que enviando la información por dirección *broadcast* se recibían con éxito un 60% de los paquetes, por ello se implemento un envío directo a *dirección ip* con el cual se conseguían transmitir prácticamente el 100% de los paquetes recibidos del UAV.

- Desarrollo de los objetivos proyectados con soluciones de bajo coste.

Por último, uno de los puntos importantes del desarrollo de la gestión de la telemetría y el control de la antena ha sido la capacidad de integrar todas estas funciones utilizando soluciones hardware de bajo coste.

A partir costes de los componentes hardware expuestos anteriormente en el Capítulo 4.2, se pretende mostrar que la solución adoptada efectivamente puede considerarse una solución de bajo coste. En primer lugar se utiliza una placa computadora Raspberry Pi como nuestra unidad central de procesos la cual se encuentra en el mercedo entorno a los 40 €. El dispositivo para la lectura de la orientación de la antena, la brújula, alcanza un coste de 15 €. El Router encargado de encaminar los paquetes por wifi está valorado en 16€. Los servomotores que generan los movimientos rotativos de la antena se encuentran en el mercado por 82€ cada uno siendo el componente más caro de los mencionados.

A falta de añadir el precio que involucra el implementar la estructura articulada en la antena de alta ganancia, el presupuesto aproximado no superaría 285 €. Este presupuesto posiblemente no cubriría el precio de otro micro-controlador convencional encargado de soportar la implementación software, pero gracias a que nos hemos ajustado a las capacidades técnicas de nuestros elementos hardware para implementar nuestra aplicación, se ha conseguido desarrollar satisfactoriamente los objetivos propuestos con una solución de bajo coste.

Bibliografía

1. The first air raid - by balloons. [Online] 2013. http://www.ctie.monash.edu/hargrave/rpav_home.html#Beginnings.

2. **Taylor, John W. R.** *Jane's pocket book of remotely piloted vehicles: robot aircraft today.* 1977.

3. **Pearson, Lee.** Developing the Flying Bomb. [Online] http://www.history.navy.mil/download/ww1-10.pdf.

4. **Bug, Kettering.** The Encyclopedia of Sciences. [Online] http://www.daviddarling.info/encyclopedia/K/Kettering_Bug.html.

5. **Fahrney, Delmar S.** *"The Birth of Guided Missiles" United States Naval Institute Proceedings December 1980.*

6. **Goebel, Greg.** Unmanned Aerial Vehicles: USA. *The road to endurance UAVs.* [Online] 2012. http://www.vectorsite.net/twuav_06.html.

7. **Wagner, W.** *Fireflies and other UAV's.* 1992.

8. A Short History of Unmanned Aerial Vehicles (UAVs). [Online] http://www.draganfly.com/news/2009/03/04/a-short-history-of-unmanned-aerial-vehicles-uavs/.

9. **Gal-Or, Benjamin.** *Vectored propulsion, supermaneuverability, and robot aircraft.* 1990.

10. Unmmanned Aerial Vehicles: USA. [Online] http://www.vectorsite.net/twuav.html.

11. General Atomics Predator. [Online] http://spyflight.co.uk/Predator.htm.

12. **Jonathan Karp, Andy Pasztor.** Drones in Domestic Skies? [Online] 2006. http://online.wsj.com/article/SB115491642950528436.html?mod=hps_us_editors_picks.

13. Unmanned plane finds child sex abuse suspect. [Online] 2007. http://www.cnn.com/2007/US/03/22/plane.border/index.html.

14. **Robinson, Amy.** FAA authorizes Predators to seek survivors. [Online] 2006. http://www.af.mil/news/story.asp?storyID=123024467.

15. Terrorists Develop Unmanned Aerial Vehicles. [Online] http://www.armscontrol.ru/UAV/mirsad1.htm.

16. **Carden, F., R. P. Jedlicka, and R. Henry.** *Telemetry Systems Engineering.* s.l. : Artech House Inc., 2002.

17. **Mayo-Wells.** *The Origins of Space Telemetry.* s.l. : Technology and Culture, 1963.

18. *Trends in Missile and Space Radio Telemetry.* **Muehlner, Joachim &.**

19. National Aeronautics and Space Administration. [Online] http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1964-077A.

20. **Molotov E.L., Nazemnye.** *Radiotekhnicheskie Sistemy Upravleniya Kosmicheskiymi Apparatami.*

21. Historia de la Raspberry Pi. [Online] http://www.bbc.co.uk/news/technology-17190918.

22. Sitio Oficial Raspberry Pi - FAQ. [Online] http://www.raspberrypi.org/faqs.

23. **Carson, Mary Kay.** *Alexander Graham Bell: Giving Voice To The World (chapter 8).* 2007.

24. **Story, Alfred Thomas.** *A story of wireless telegraphy.* 1904.

25. What is Wi-Fi? [Online] http://www.webopedia.com/TERM/W/Wi_Fi.html.

26. A brief history of Wi-F. [Online] http://www.economist.com/node/2724397.

27. **Ritchie., Dennis M.** *The Development of the C Language. History of Programming Languages.* 1996.

28. **Tucker, Allen B.** *Lenguajes de Programación.* s.l. : McGraw-Hill México, 1992.

29. The Making of Python. [Online] http://www.artima.com/intv/pythonP.html.

30. The History of HTML. [Online] http://www.ironspider.ca/webdesign101/htmlhistory.html.

31. **Nelson, Robert A.** *The Global Positioning System.* s.l. : Via Satellite, 1999.

32. **Ducard, Dr. G.** Unmanned Aircraft Design, Modeling and Control. *Introduction to Navigation Systems.*

33. **Benet, G.** Transparencias dispositivos industriales. *Tema1: Transductores.* 2012.

34. **Torre, Marcos Ávila de la.** Sensores de velocidad. 2012.

CONTENIDO ANEXO A: MANUAL DE PROGRAMACIÓN

1. INTRODUCCIÓN

En este manual se pretende explicar de forma detallada cada una de la las funciones del código desarrollado con el fin de permitir que otro programador con experiencia pueda entender, modificar e incluso replicar el código de nuestra aplicación.

El programa tiene como finalidad gestionar los recursos hardware de nuestro sistema que van interconectados a nuestra placa computadora y base de programación Raspberry Pi. El código está desarrollado en lenguaje C en el entorno del sistema operativo Linux bajo la herramienta IDE libre "Geany". Es compilado con el compilador gcc que viene por defecto instalado en la Raspberry Pi, y a la hora de ejecutarlo se cuenta con un procesador con arquitectura ARM con capacidad suficiente para correr nuestra aplicación de gestión de telemetría y control de la antena.

De todas las funciones elaboradas tanto para los hilos de ejecución como las desarrolladas en nuestra librería, podemos decir que prácticamente todo el código está diseñado para una aplicación específica, es decir, diseño *ad hoc*. Por ello además de elaborar la memoria del proyecto en donde no se explica con detalle el funcionamiento de cada función, se ha considerado anexar este manual de programación.

1.1. Estructura del programa

Para facilitar el seguimiento de la explicación de código, primero recordaremos el funcionamiento básico tanto del programa principal como de los hilos independientes que gestionan los distintos subsistemas de nuestra aplicación.

Nuestro software, como se mencionaba en el Capítulo 5, consta de una función principal *main* y cinco hilos de ejecución independientes. A continuación se refrescará la función principal de cada uno de ellos para luego pasar a la explicación de su funcionamiento en mayor detalle.

- **Main (Función principal):** Es la función desde la que se inicializan y llaman los cinco hilos. Si esta función terminase el resto de los hilos terminarían de ejecutarse también, por ello la programación está realizada de modo que esta función *main* nunca termine hasta el caso en que los cinco hilos acabasen de

ejecutar. Además de gestionar los hilos, en la función principal se gestionan los argumentos de entrada por línea de ejecución del programa que luego se pasaran como parámetro en variable global a los hilos que lo requieran.

- **Telemetry_SerialPortManagement:** La función de este hilo forma parte de la implementación de la gestión de la telemetría que nos permite recibir y enviar datos del avión. Aquí se configura y pone en marcha la comunicación por protocolo puerto serie entre la Raspberry Pi, encargada de gestionar todos los dispositivos, y el módulo de Radio Enlace que se comunicará directamente con el UAV emitiendo y recibiendo a una frecuencia de 2,45GHz. Para ello, la función se encargará de abrir el socket necesario para la comunicación, establecer la velocidad de lectura en baudios, y configurar las líneas de lectura /escritura haciendo que sean no bloqueantes en caso de que no pudiesen leer o escribir nada.

- **Telemetry_UDPManagement:** Este hilo concierne la segunda función que interviene en la gestión de la telemetría y corresponde a la siguiente parte de la transmisión que comunica de nuevo la Raspberry Pi a través de un Router Wifi con los Puntos de monitorización que se conecten a nuestra red local Wifi. Hay que destacar que en el proceso de repetir la información proveniente del UAV se realiza mediante protocolo UDP (User Datagram Protol). Mediante este sistema de envío no se necesita tener una conexión previa con el receptor como ocurre con el protocolo TCP/IP.

- **Antenna:** La finalidad de este hilo es, una vez gestionada la telemetría, utilizar correctamente los datos recibidos de la plataforma de vuelo para conseguir que la antena con dos grados de libertad rotativos apunte continuamente al avión. Entre las tramas recibidas se encuentran los datos de latitud, longitud y altura del UAV con respecto a la superficie de la tierra. Mediante una serie de aproximaciones y cambios de sistema de referencia se obtienen el ángulo de azimut que gobierna el movimiento rotativo sobre el eje vertical de la antena, y el ángulo de inclinación que varía con la altura a la que se encuentra el UAV.

- **UAV_scanning:** El objetivo de este hilo es mejorar la robustez del sistema. Para ello lanza una función de rastreo se activa de forma automática en el momento que se pierde la señal del avión. Se trata de una función que se arranca desde la inicialización del programa, pero que se encuentra a la espera de que se dejen de recibir datos válidos de localización por la telemetría para comenzar a funcionar. De este modo, en el momento que el avión deja de ser localizado, la antena empieza a realizar un barrido a velocidad intermedia y no para hasta recibir de nuevo un dato válido lo que implica que el UAV ha sido encontrado de nuevo.

- **Compass_reading:** El último hilo gestiona de forma independiente la lectura de la orientación de la antena en tiempo real. Para hacer el sistema más robusto una vez más, se utiliza una brújula digital solidaria a la antena que entrega en todo momento el ángulo global de esta. A la hora de localizar el avión es necesario referenciar la posición de la antena. Por ello a la hora de inicializar el proceso de control de los movimientos rotativos de la antena es necesario leer el ángulo que nos da la brújula en el instante inicial lo cual gestiona la orientación que tiene la antena de forma autónoma. Por ello es necesario lanzar un hilo independiente que este continuamente leyendo y guardando los valores de la brújula en una variable global de modo que ese valor esté disponible en tiempo real para el resto de funciones que lo necesiten.

Para recordar una vez más las interconexiones y los niveles de comunicación que intervienen entre la función principal y los cinco hilos de ejecución se muestra un esquema en el que se diferencia la funcionalidad de cada una de los hilos.

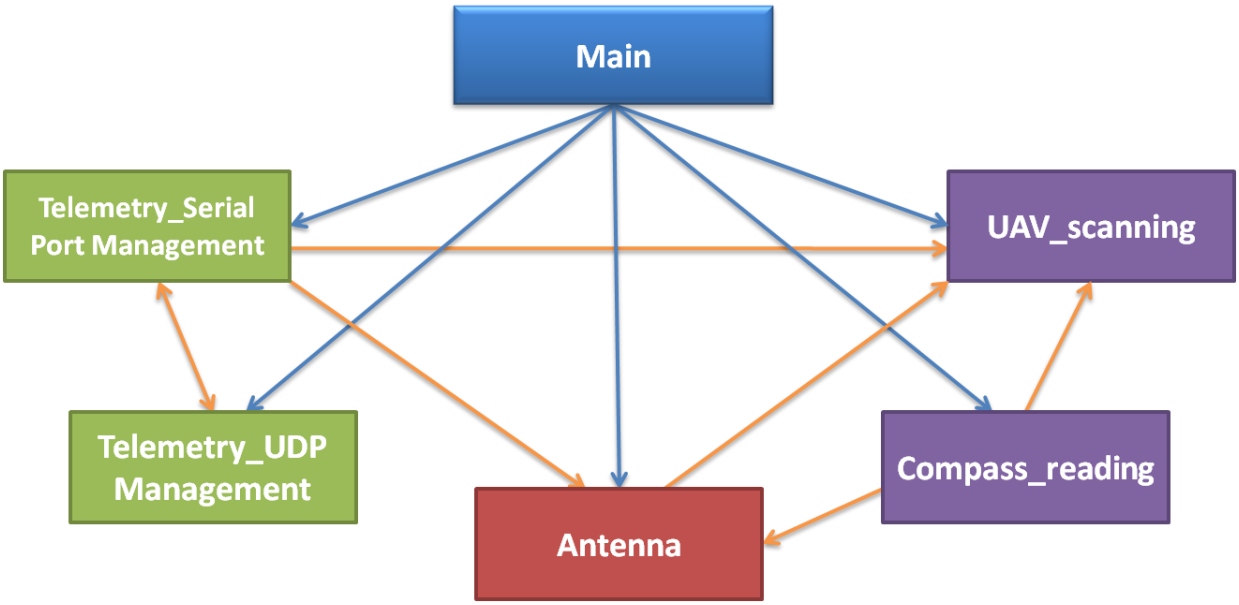

Figura 39. Función principal del sistema e hilos de ejecución

En la figura se muestra la función *Main* que se encarga de llamar al resto al principio y no intercambia ninguna información más durante la ejecución.

Las funciones sombreadas en verde son las encargadas de gestionar la telemetría e intercomunicarse entre ellas para crear los flujos de paquetes entre el UAV y es sistema de monitorización. *Telemetry_Serial Port Management* intercambiará información con el hilo que gestiona el control de la antena. Esta comunicación será de un solo sentido ya que únicamente se enviarán los datos de localización recibidos por la telemetría hacía el control de los servos. También se encargará de avisar a la función de rastreo de señal cuando debe comenzar a buscar.

La función *Antenna* en rojo es la única encargada de gestionar el control de la base robotizada de la antena a partir de los parámetros recibidos de la telemetría. También está comunicada con los hilos de la función de rastreo al inicio del programa, y con la lectura de la brújula.

Por su parte la función *UAV_scanning* encargada de facilitar la búsqueda de señal cuando el UAV se encuentra perdido, recibe órdenes de cuando debe empezar a buscar de la telemetría. También debe esperar al principio a que el control de la antena este correctamente inicializado, de ahí la flecha en un solo sentido con la

función *Antenna*. Y además recibe información continua sobre la orientación de la antena recogida en la función que lee la brújula.

Por último el hilo *Compass_reading* se encarga de leer de forma continua la brújula digital y almacenar el valor de la orientación de la antena en una variable global que compartirá con *Antenna* y *UAV_Scanning.*

2. DEFINICIONES Y VARIABLES GLOBALES DEL PROGRAMA

Antes de comenzar a explicar en detalle el funcionamiento de cada subsistema, conviene conocer cuáles son las variables globales de nuestro programa. Con ellas se va a conseguir sincronizar los diferentes hilos entre sí, e intercambiar información de necesaria accesibilidad por más de un hilo a la vez. Además de variables, también se exponen las definiciones creadas con motivo de facilitar la programación y hacer un código más intuitivo tanto para el programador como para el lector.

2.1. Definiciones globales

En primer se exponen las definiciones *(#defines y typedefs*) implementados para la mejor comprensión del código:

- Por un lado se definen los puertos que van a ser utilizados en la comunicación UDP tanto para enviar como para recibir paquetes.

```
//------------------------------------------------------------
//Port's defines for UDP transmission
//------------------------------------------------------------
#define PORT_send 5557
#define PORT_send_orientation 5558
#define PORT_receive 9931
```

Código fuente 1. Definiciones globales de puertos

- También se definen los parámetros de entrada de la función que leerá la brújula digital para que esta nos devuelva el parámetro deseado (yaw, pitch y roll). En nuestro caso únicamente leeremos el yaw que será nuestro ángulo de referencia.

```
//---------------------------------------------------------
//Compass commands definitions
//---------------------------------------------------------
#define GET_ANGLE_16 0x13
#define GET_PITCH 0x14
#define GET_ROLL 0x15
```

Código fuente 2. Definiciones globales para la lectura de la brújula

- También se han definido las rutas de lectura por comunicación serie con la brújula digital y con el controlador de los servos. La brújula es accedida por los pines del GPIO y los servos por una de las salidas USB de la Raspberry Pi.

```
//---------------------------------------------------------
//Serial ports root definition
//---------------------------------------------------------

//GPIO Serial port. Compass
#define s_port_compass "/dev/ttyAMA0"

//USB-serial port. Antena
#define s_port_ant "/dev/serial/by-path/platform-bcm2708_usb-
usb-0:1.2:1.0-port0"
```

Código fuente 3. Definiciones globales de rutas para comunicación serie

- Por otro lado se ha hecho uso de la función *typedef* para definir un nuevo tipo de variables específicas para algunas de nuestras funciones. Definimos una variable de tipo *unsigned char* utilizada para el tratamiento de bytes, y también una serie de *flags* como booleanos que se explican en el apartado 2.2.

```
/*Global variables for Antenna and Compass management*/
typedef unsigned char byte;


/*Global variable for UAV_scanning function*/
typedef enum {found, lost} bool1;
typedef enum {obtained, searching} bool2;
typedef enum {right, left} bool3;
```

Código fuente 4. Definiciones de nuevos tipos de variables

2.2. Variables globales

En este apartado pasamos a exponer y explicar cuáles son las variables globales utilizadas en el programa.

- En primer lugar, la estructura global definida para albergar los datos de los ángulos de Euler del avión y su localización. Esta estructura será escrita en la lectura de datos válidos del radio enlace, y será accedida por el hilo que controla los movimientos de la antena para seguir al UAV.

```
/*Global variable for the UAV position and orientation data*/
struct Parameters
{
            float roll;
            float pitch;
            float yaw;
            double latitude;
            double longitude;
            double altitude;
};
```

Código fuente 5. Variable global: Estructura para datos de localización del UAV

- Variables globales especificas para la telemetría. Utilizaremos cuatro buffers globales y tres punteros a tres de dichos buffers para la correcta sincronización entre el hilo que gestiona la comunicación serie con el módulo de radio y el hilo que se encarga de la transmisión por UDP.

```
/*Global variable for the Telemetry*/
int pointer_bufUDP=0, received_udp=0;
unsigned char buf_SendUDP[1000], buf_PC[1000];
unsigned char buf_SendUDP_orientation[50];

/*Global variables for the Radio Link reading*/
unsigned char data_AHRS[5000];
int data_pointer=0;
```

Código fuente 6. Variable global: Buffers para la telemetría

- Variables para la conexión de varios puntos de monitorización a nuestro sistema. Para poder conectar varios ordenadores a nuestra red local wifi, se van almacenando en un array global las direcciones de estos para más tarde poder abrir los respectivos socket necesarios para la comunicación.

```
/*Global variables for the UDP sending*/
char ip_address[3][50];
int num_ipaddress=0;
```

Código fuente 7. Variable global: Array de IP para la conexión wifi

- Variables de la gestión del control de la antena y la lectura de la brújula. Para poder comprobar si las comunicaciones con el controlador de la antena y con la brújula digital fueron satisfactoriamente abiertas es necesario colocar el manejador de estas conexiones en una variable global que pueda ser accedida por más de un hilo del sistema que requiere de esta comprobación. Por otro lado también es de utilidad tener la lectura de la brújula y la localización inicial de la antena en una variable global para su lectura desde distintos hilos.

```
/*Global variables for Antenna and Compass management*/
int socket_antenna=-1;
int socket_compass=-1;
double global_compass=-1;
double origin[3];
```

Código fuente 8. Variable global: Control de la antena y acceso a la brújula

- Variables globales para la función de rastreo. Por último, se han declarado una serie de variables globales necesarias para el correcto funcionamiento de la función de búsqueda en caso de pérdida del avión. En primer lugar tenemos una serie de *flags* definidos como booleanos que nos indicarán si el avión se encuentra perdido o encontrado, si al inicio del programa se ha establecido correctamente el ángulo azimut de referencia, y si en la sesión de búsqueda la antena se encontrada barriendo a izquierdas o a derechas para predecir la

trayectoria del avión. En segundo lugar se tiene una variable que cuenta los paquetes fallidos recibidos por el radio enlace y que sirve para gestionar cuando se empezará a realizar la búsqueda. Además se para establecer la inercia anteriormente explicada, se requiere de una variable global que vaya guardando continuamente la lectura "n-1" de la brújula para luego compararla con la actual y un contador que permite establecer un intervalo entre comparación y comparación. Por último, las últimas dos variables globales se refieren al umbral de tiempo que queremos que pase antes de que la función *scan_signal* comience a funcionar y la velocidad a la que deseamos que rote la antena durante dicha búsqueda.

```
/*Global variable for UAV_scanning function*/
bool1 flag_control=lost;
bool2 flag_azimut_inicial=searching;
bool3 flag_antenna_inertia=right;

int fail_counter_RadioLink=0;

double old_compass=0.0;
int times=0.0;

int scanning_threshold=0;
int scanning_speed=50;
```

Código fuente 9. Variable global: Función de rastreo de señal del UAV

3. FUNCIÓN PRINCIPAL DEL PROGRAMA

La función principal del programa es la primera que se arranca para ejecutar nuestro programa y es la encargada de inicializar y lanzar el resto de hilos de ejecución. Para poner en marcha estos threads se se utiliza la siguiente secuencia:

1) inicialización *pthread_t* y el identificador.
2) Seguidamente, la función de *pthread_create* enlanzando el hilo creado con el puntero a la función que queremos que este ejecute. Como se muestra en el codigo fuente 10 de la funcion *main*, en nuestro programa no se pasan argumentos de entrada a las funciones de los hilos ya que la comunicación entre ellos se produce mediante las variables globales que se explicaron en el apartado anterior. Por ello los campos marcados como “NULL”.
3) Una vez inicializados y ejecutados los cinco threads, se utiliza la sentencia *pthread_join* con el identificador del thread como argumento y un puntero al lugar donde queremos que se guarde el estado a la finalización de este thread y que lo rellenaremos con “NULL” en nuetro caso. De este modo se consigue que la función principal se quede suspensa hasta que todos los hilos finalicen.

Otra tarea de la que se encarga nuestra función principal es de recoger parametros de entrada por la linea de comandos en su ejecución. Estos parametros serán introducidos por el servidor web y serán escritos a continuación de la linea *./main* de la consola del sistema operativo. La función de nuestra librería encargada de esta tarea se llama *InputParametersManagement* y como parametros de entrada tendrá el número de argumentos recibidos y un array de strings con los argumentos. La gestión realizada por esta función sigue los sieguiente pasos:

1) Comprobar que el número de argumentos de entrada sean 8, en caso contrario se emitirá un error.
2) Convertir los tres primeros argumentos que corresponden a la latitud, longitud y altura de la antena en parametros de tipo “double float”, y los dos parametros de la función rastreo en variables de tipo entero.
3) Comprobar si los últimos tres argumentos de entrada correspondientes a las direcciones IP de los puntos de monitorización que desean conectarse a

nuestra red local son válidos (del tipo 192.168.2.X) y dentro del rango Xϵ[100,254]. En caso de que ninguna de las direcciones sea válida, se dará asignará automaticamente el valor de la dirección de envío por *broadcast* (192.168.2.255) para repetir los paquetes a cualquier destinatario de la red.

A continuación se muestra el código fuente de la función principal en donde se pueden seguir los pasos explicados:

```
int main(int argc, char *argv[])
{
	/*Collecting input arguments from the WebServer*/
	InputParametersManagement(argc, argv);

	/*Execution of the differents threads in the program*/
	pthread_t thread_sensor;
	pthread_create(&thread_sensor,NULL,Compass_reading,NULL);

	pthread_t thread_telemetry_SerialPortManagement;
	pthread_create(&thread_telemetry_SerialPortManagement,NULL,
			Telemetry_SerialPortManagement,NULL);

	pthread_t thread_telemetry_UDPManagement;
	pthread_create(&thread_telemetry_UDPManagement,NULL,
			Telemetry_UDPManagement,NULL);

	pthread_t thread_antenna;
	pthread_create(&thread_antenna,NULL,Antenna,NULL);

	pthread_t thread_scanning;
	pthread_create(&thread_scanning,NULL,UAV_scanning,NULL);


	/*main() cannot finish unless every thread is finished*/
	pthread_join(thread_telemetry_UDPManagement,NULL);
	pthread_join(thread_telemetry_SerialPortManagement,NULL);
	pthread_join(thread_scanning,NULL);
	pthread_join(thread_antenna,NULL);
	pthread_join(thread_sensor,NULL);

	return 0;
}
```

Código fuente 10. Función principal del programa

4. HILOS DE EJECUCIÓN INDEPENDIENTE

En este apartado se analiza en detalle los cinco hilos de ejecución de los que consta el programa tal y como se hizo con la función principal.

4.1. Thread de gestión de comunicación serie en la telemetría

El hilo de gestión de la comunicación puerto serie se encarga, como y se adelanto en la introducción de este manual, de organizar las transferencia de datos entre las Raspberry Pi y el módulo de radio (Radio Link) destinado a servir a la telemetría de nuestro sistema.

A continuación se explica que pasos sigue esta función para trabajar, y seguidamente se muestra el fragmento del código explicado:

1) Se inicializan una serie de variables locales que serán útiles dentro de nuestra función además del uso de alguna de las variables globales ya explicadas.
2) Se abre una conexión serie con la función creada *OpenSerialPort_Telemetry*. Dentro de esta se configura una comunicación de lectura y escritura no bloqueante en caso de no haber paquetes en el buffer. Se establece una velocidad de transferencia de 19200 baudios y 8 bits de lectura. El resto de configuraciones se mantienen por defecto.
3) Una vez abierto el socket de comunicación serie correctamente se entra en el bucle infinito que gobierna este hilo.
4) Dentro del bucle va a tomar el tiempo del reloj interno justo antes y después de ejecutar todas las instrucciones dentro de este, para de este modo poder fijar el tiempo de ciclo deseado con la resta de estos muestreos de tiempo y la función *usleep()*.
5) A continuación procedemos a leer información del radio enlace con la función *Read_RadioLink* en un proceso que se explica en detalle en apartado 0.
6) En el caso de leer alguna información, se procederá a almacenar esta información en otro buffer para su posterior envío por UDP. Para la correcta sincronización de ambos buffers es necesario llevar el control de un puntero del buffer *buf_SendUDP* en donde se va a copiar la información. Para gestionar correctamente este puntero es imprescindible que en el momento

de acceso a este, ningún otro hilo pueda modificarlo. Para ello se hace uso de la estrategia MUTEX de exclusión entre threads.

```
void *Telemetry_SerialPortManagement()
{
	int h=0;
	int read_sp=0, written_sp=0;
	int handler_sp=-1;
	unsigned char buf_RadioLink[1000];
	struct timeval t1_ps, t2_ps, dt_ps;

	OpenSerialPort_Telemetry(&handler_sp);

	while(handler_sp>0)
	{
		gettimeofday(&t1_ps,NULL);

		Read_RadioLink(handler_sp,  &data,  buf_RadioLink,
		&read_sp);

		if(read_sp>0)
		{
			pthread_mutex_lock(&lock_pointerUDP);

			for(h=0;h<read_sp;h++)
			{
				buf_SendUDP[pointer_bufUDP+h]=
				buf_RadioLink[h];
			}
			pointer_bufUDP+=read_sp;

			pthread_mutex_unlock(&lock_pointerUDP);
		}
```

Código fuente 11. Hilo de gestión comunicación serie para la telemetría (Parte 1)

Una vez hemos explicado uno de los sentidos de comunicación, del RadioLink a la Raspbery Pi, queda describir el sentido contrario que conlleva escribir por el RadioLink los datos que nos llegarán de la transmisión UDP.

Puesto que es una continuación de la secuencia del código que acabamos de explicar en un sentido, vamos a seguir enumerando los pasos como corresponde:

7) Comprobamos si hay datos en el buffer recibidos por la comunicación UDP.

8) Si hay algo, escribimos esta información con el identificador de la comunicación serie abierta en el código anterior.
9) Volvemos a leer el tiempo de reloj en ese momento y gracias a una estructura de tipo *timeval* inicializada al principio de la función de este hilo, podremos calcular el intervalo de tiempo que a transcurrido en la ejecución del código y restárselo al tiempo que deseamos establecer como tiempo de ciclo que en nuestro caso para el Thread de la gestión de la comunicación serie para la telemetría será de 25 milisegundos.
10) Por último, queda detallar que en caso de que el bucle no se iniciará correctamente, se cerraría el socket creado para la comunicación serie y se suspendería el hilo con la función *pthread_exit*.

```
            if(received_udp>0)
            {
                  Write_RadioLink(handler_sp, received_udp,
                                    &written_sp, buf_PC);

                  received_udp=0;
            }

            gettimeofday(&t2_ps,NULL);
            dt_ps.tv_sec=t2_ps.tv_sec-t1_ps.tv_sec;
            dt_ps.tv_usec=t2_ps.tv_usec-t1_ps.tv_usec;
            usleep(25000-dt_ps.tv_usec);
      }

      close(handler_sp);
      pthread_exit(NULL);
}
```

Código fuente 12. Hilo de gestión comunicación serie para la telemetría (Parte 2)

4.1.1. Función *Leer del Radio Link*

Esta función carga sobre un vector los bytes almacenados en el buffer del puerto serie correspondiente, y busca en dicho vector una trama entera, empezando por detrás. Al empezar por detrás nos aseguramos de quedarnos siempre con los valores más actualizados. Si se encuentra una trama entera, esta se decodifica y los valores de la lectura se almacenan en una variable global llamada *data* estructura que se compartirá con el hilo encargado de la gestión del control de la antena. Por otro lado, la parte del buffer anterior a la trama encontrada se desecha y la parte posterior se guarda para ser concatenada al comienzo de la siguiente lectura.

Si por el contrario no se encuentra bloque alguno, se guarda el vector completo para concatenarlo en la siguiente repetición al inicio del buffer. Algunos trozos de esta función se adjuntan a continuación.

El código fuente del cuerpo de la función *Read_RadioLink* se muestra a continuación:

```
if (bytes_read>0)
{
	for(h=0;h<bytes_read;h++)
	{
		data_AHRS[data_pointer+h]=buffer[h];
		msg1[h]=buffer[h];
	}
	av_bytes=data_pointer+bytes_read;

	afterHeaderPos=-1;
	endPosition=-1;
	t=0;
	for(j=av_bytes-1;j>=2;j--){
		if((data_AHRS[j-2]==0x03)&&
		(data_AHRS[j-1]==0xFF) &&
		(data_AHRS[j]==0x02)&&(t==0))
		{	t=1;
			endPosition=j-2;
			j=j-2;
		}
		if((data_AHRS[j-2]==0x03) &&
		(data_AHRS[j-1]==0xFF) &&
		(data_AHRS[j]==0x02)&&(t==1))
		{	afterHeaderPos=j+1;
			break;
		}
	}
```

Código fuente 13. Función Read_RadioLink (Parte 1)

```
      if((afterHeaderPos!=-1)&&(endPosition!=-1))
      {
            n=endPosition-afterHeaderPos;
            CRC_calc=calcCRC((unsigned char*)
            &data_AHRS[afterHeaderPos], n-2);
            CRC=(data_AHRS[endPosition-2]<<8) |
            data_AHRS[endPosition-1];

            if (CRC==CRC_calc)
            {
                  fail_counter_RadioLink=0;
                  flag_control=found;


                  data->roll=floatFromBytesBE((unsigned
                  char*)&data_AHRS, afterHeaderPos+3);
                  data->pitch=floatFromBytesBE((unsigned
                  char*)&data_AHRS,afterHeaderPos+7);
                  data->yaw=floatFromBytesBE((unsigned
                  char*)&data_AHRS,afterHeaderPos+11);

                  data->latitude=doubleFromBytesBE((unsigned
                  char*)&data_AHRS,afterHeaderPos+15);
                  data->longitude=doubleFromBytesBE((unsigned
                  char*)&data_AHRS,afterHeaderPos+23);
                  data->altitude=doubleFromBytesBE((unsigned
                  char*)&data_AHRS,afterHeaderPos+31);
            }

            data_pointer=av_bytes-endPosition;
            for (i=0;i<=data_pointer;i++)
            {
                  data_AHRS[i]=data_AHRS[endPosition+i];
            }
      }
      else
            data_pointer+=bytes_read;
}
else if(bytes_read<=0)
{
      fail_counter_RadioLink++;
      if(fail_counter_RadioLink>scanning_threshold)
            flag_control=lost;
}
```

Código fuente 14. Función Read_RadioLink (Parte 2)

4.2.Thread de gestión de comunicación por UDP en la telemetría

Este hilo de gestión de la comunicación por UDP tiene como finalidad comunicar nuestra antena con cualquier sistema de monitorización que sea capaz de conectarse a nuestra red local por wifi.

La gestión de la telemetría está separada en dos hilos que trabajan de manera independiente pero sincronizada, con el fin de conseguir una estructura del programa más modular y poder configurar diferentes tiempos de ciclo según lo requiere la frecuencia de lectura del Radio Link y el envío de paquetes por UDP. En nuestro caso era interesante tener un periodo de muestreo de 25 milisegundos para el hilo de la comunicación serie (apartado 4.1) y por el contrario no era conveniente repetir paquetes por UDP cada menos de 100 milisegundos, por ello el principal argumento para esta programación modular. La sincronización entre estos dos hilos se realiza esencialmente gracias a los buffers declarados como variables globales descritos en el apartado 2.2.

En resumen, esta función lo que consigue es enviar por protocolo UDP las tramas que vienen directamente del radio enlace y también un paquete de datos que contiene la localización y orientación constante de la antena.

Al igual que con la función de comunicación serie, se explica la secuencia de pasos de operación seguida del fragmento del código del thread:

1) En primer lugar se declaran todas las variables locales a utilizar por la función, entre ellas, tres arrays de tiempo estructura *sockaddr_in* para albergar la información de los datos de la comunicación vía UDP.
2) Una vez declaradas, se inicializarán dichos atributos de estas estructuras según el número de clientes que vayan a recibir paquetes. Con un bucle “for” y la variable global *num_ipaddress,* que contiene cuantas direcciones válidas fueron introducidas en el servidor web, se inicializan dichos atributos con las direcciones de cada cliente y el puerto de comunicación. En el fragmento de *código fuente 15* se muestran estos dos primeros puntos.

```
void *Telemetry_UDPManagement()
{
	int i=0;
	int sent_udp=0, sent_udp_orientation=0;
	int  handler_sendUDP[3],  handler_sendUDP_orientation[3],
	handler_receiveUDP=-1;

	struct timeval t1_udp, t2_udp, dt_udp;

	struct  sockaddr_in  sender[3],  sender_orientation[3],
	receiver, receiver_recv;

	for(i=0;i<num_ipaddress;i++)
	{
		bzero(&sender[i], sizeof(sender[i]));
		sender[i].sin_family = AF_INET;
		sender[i].sin_port = htons(PORT_send);
		sender[i].sin_addr.s_addr =
		inet_addr(ip_address[i]);

		bzero(&sender_orientation[i],
		sizeof(sender_orientation[i]));
		sender_orientation[i].sin_family = AF_INET;
		sender_orientation[i].sin_port =
		htons(PORT_send_orientation);
		sender_orientation[i].sin_addr.s_addr =
		inet_addr(ip_address[i]);
	}

	bzero(&receiver, sizeof(receiver));
	receiver.sin_family = AF_INET;
	receiver.sin_port = htons(PORT_receive);
	receiver.sin_addr.s_addr = htonl(INADDR_ANY);
```

Código fuente 15. Hilo de gestión comunicación UDP para la telemetría (Parte 1)

3) El siguiente paso es abrir los socket necesarios para dichas comunicaciones antes de entrar en el bucle de envío y recepción. Al igual que la inicialización, con un bucle “for” y el número de clientes a conectar, se abren dos socket de envió por cada cliente: Uno para reenviar las tramas y otro para la orientación de la antena.
4) A continuación se abre también un socket para en este caso recibir posibles paquetes para las acciones de control enviadas al avión.

```
for(i=0;i<num_ipaddress;i++)
{
      OpenSocketSending_Telemetry(&handler_sendUDP[i],
      &sender[i]);

      OpenSocketSending_Telemetry(
      &handler_sendUDP_orientation[i],
      &sender_orientation[i]);
 }

 OpenSocketReceiving_Telemetry(&handler_receiveUDP,
 &receiver);
```

Código fuente 16. Hilo de gestión comunicación UDP para la telemetría (Parte 2)

5) Una vez realizadas correctamente todas las inicializaciones anteriores se entra en el bucle que enviará y recibirá de manera cíclica paquetes de datos por UDP. Nada más empezar se lee el tiempo de reloj como también se hacía en la comunicación serie para al final gestionar el tiempo de ciclo.
6) Al sincronizar este thread con el anterior 4.1, vuelve a ser necesario el uso de la estrategia MUTEX de exclusión mutua entre hilos. Se bloque y se desbloquea el acceso a las variables globales entre threads durante las líneas que lo requieren.
7) Se comprueba si hay información acumulada de el buffer *buf_SendUDP* y de ser así se reenvía esta información a los clientes por UDP con la función *Forward_RadioLink_UDP.* Una vez enviada, se vacía el buffer poniendo a cero el puntero de apilamiento del mismo.
8) Se desbloquea la estrategia MUTEX y se reenvía de nuevo la información, esta vez con la orientación y localización de la antena a los clientes conectados que monitorizarán la telemetría.
9) El último paso en la comunicación es leer el buffer UDP por si hubiese posibles paquetes con acciones de control y guardarlas en nuestro buffer *buf_PC*.
10) Como hacíamos el thread 4.1, volvemos a leer el tiempo del reloj interno y con la diferencia de tiempos establecemos esta vez un periodo de ciclo de 100 milisegundos con la función *usleep()*.

```
	while( (handler_sendUDP[0]>0) && (handler_receiveUDP>0)
	&& (handler_sendUDP_orientation[0]>0))
	{
		gettimeofday(&t1_udp,NULL);

		pthread_mutex_lock(&lock_pointerUDP);

		if(pointer_bufUDP>0)
		{
			for(i=0;i<num_ipaddress;i++)
				Forward_RadioLink_UDP(
				handler_sendUDP[i], pointer_bufUDP,
				&sent_udp, buf_SendUDP, &sender[i]);

			pointer_bufUDP=0;
		}
		pthread_mutex_unlock(&lock_pointerUDP);

		if((strlen(buf_SendUDP_orientation))>0)
		{
			for(i=0;i<num_ipaddress;i++)
				Forward_RadioLink_UDP(
				handler_sendUDP_orientation[i], 32,
				&sent_udp_orientation,
				buf_SendUDP_orientation,
				&sender_orientation[i]);
		}

		Read_PC_UDP(handler_receiveUDP, &received_udp,
		buf_PC, &receiver_recv);

		gettimeofday(&t2_udp,NULL);
		dt_udp.tv_sec=t2_udp.tv_sec-t1_udp.tv_sec;
		dt_udp.tv_usec=t2_udp.tv_usec-t1_udp.tv_usec;
		usleep(100000-dt_udp.tv_usec);
	}
	pthread_exit(NULL);
}
```

Código fuente 17. Hilo de gestión comunicación UDP para la telemetría (Parte 3)

4.3.Thread de gestión del control de la antena

Este hilo es el encargado de gestionar prácticamente todas las funciones necesarias, para lograr la autonomía de la antena. En este hilo de ejecución se llevan a cabo las siguientes tareas:

1) Definición de variables locales

```
double initial_azimuth;
int p0, p1, speed=150;
double x=0,y=0,z=0;
double azimuth=0, elevation=0;
double origin_lat=origin[0], origin_long=origin[1],
origin_alt=origin[2];
```

Código fuente 18. Hilo de gestión del control de la antena: Definición de variables locales

Estas variables son las que se usan para los procesos de cálculo de la orientación de la antena. Se parte de la variable global *data*, en la cual se guarda la posición en coordenadas GPS del avión, y tras el proceso de cálculo se obtiene el valor de las variables *p0* y *p1*, que son los parámetros que se envían al controlador de los servos de la plataforma mecánica para que ésta se mueva.

2) Apertura de los puestos serie

```
open_antenna();
open_compass();
```

Código fuente 19. Hilo de gestión del control de la antena: Apertura de los puertos

La apertura de los puestos serie para la gestión de la plataforma y de la brújula digital, se realiza llamando a las funciones que se muestran en extracto del código anterior. Estas dos funciones están implementadas en la librería *library_functions.c* y en cada una se configura el puerto serie respectivo, con su correspondiente configuración.

En la configuración del puerto serie para la comunicación con la plataforma, se usa la configuración predeterminada del sistema, salvo la velocidad del puerto, que es el único parámetro que se configura manualmente. Este valor se establece a 115200 baudios como se puede ver en el siguiente código.

```
cfsetispeed(&t_serial_antenna,B115200);
cfsetospeed(&t_serial_antenna,B115200);
```

Código fuente 20. Hilo de gestión del control de la antena: Comunicación con servos

Por otra parte, la configuración del puerto serie para la comunicación con la brújula digital, se establece el valor de muchos más parámetros como se puede ver en el siguiente extracto de código.

```
t_serial_compass.c_cflag |= B9600|CBAUDEX|CS8|CSTOPB|CLOCAL|CREAD;
t_serial_compass.c_iflag |= IGNPAR;
t_serial_compass.c_oflag = 0;
t_serial_compass.c_lflag &= ~(ICANON|ECHO|ECHOE|ISIG);
t_serial_compass.c_cc[VMIN] = 1;
```

Código fuente 21. Hilo de gestión del control de la antena: Comunicación con brújula

En la configuración anterior, se establece la velocidad del puerto a 9600 baudios, paquete de datos de 8 bits, 2 bits de stop, entre otros.

3) Obtención del ángulo inicial de referencia

El ángulo de referencia se obtiene cuando la plataforma se encuentra en su posición inicial. Para obtenerlo, se llama a una función ubicada en la librería *library_functions.c* mediante el siguiente código.

```
initial_azimuth = get_initial_azimuth();
flag_azimut_inicial=obtained;
```

Código fuente 22. Hilo de gestión del control de la antena: Obtener ángulo inicial

En la función *get_initial_azimuth()* se envía la plataforma a su posición de origen.

```
send_antenna(650, 300, 1500, 500);
```

Código fuente 23. Hilo de gestión del control de la antena: Envío antena a origen

Como no se tiene ningún sensor que indique la posición del eje de azimut de la plataforma, se hace uso de la brújula digital para conocer el momento en el que la

plataforma deja de moverse. Para ello, se realiza la varianza de los datos leídos de la propia brújula. Cuando la varianza está por debajo de cierto valor preestablecido, se considera que la plataforma ha dejado de moverse. Entonces se guarda el último valor leído de la brújula, como el ángulo inicial de referencia.

```
while (varianza>0.5){

      usleep(100000);
       compass[3]=compass[2];
       compass[2]=compass[1];
       compass[1]=compass[0];
       compass[0]=global_compass;

      varianza=varianza_muestral(compass,4);
}

start_angle=compass[0];
```

Código fuente 24. Hilo de gestión del control de la antena: Comprobación varianza

Por último, el ángulo obtenido se re-escala y se transforma, para evitar que el ángulo obtenido sea mayor que 360 o menor que 0, y para que el norte de la brújula esté sobre el eje X en lugar del eje Y.

```
initial_azimuth = 360.0 - start_angle + 90.0;
initial_azimuth = rescale_angle(initial_azimuth);
```

Código fuente 25. Hilo de gestión del control de la antena: Re-escalar ángulos

4) Obtención y envío de los parámetros de la plataforma

Como se comentó anteriormente, para obtener los parámetros de orientación que han de enviarse a la plataforma (*p0* y *p1*), se parte de los datos de posición GPS del avión almacenados en la variable global *data*. Dichos datos se transforman a coordenadas cartesianas referenciados a la posición de la antena. El resultado obtenido se transforma en coordenadas esféricas, obteniendo un ángulo de elevación y azimut

entre el avión y la antena. A partir de estos dos ángulos se calculan los parámetros de posición de los dos servomotores de la plataforma mecánica.

```
gps_xyz(origin_lat, origin_long, data.latitude, data.longitude,
  data.altitude, &x,&y,&z);

ref_cal(x, y, z, initial_azimuth, origin_alt, &azimuth, &elevation);

antenna_params_calc(azimuth,elevation, &p0, &p1);

send_antenna(p0, 300, p1, 300);
```

Código fuente 26. Hilo de gestión del control de la antena: Cálculo de parámetros

Para el cálculo de los parámetros de la plataforma, se hace una interpolación utilizando los ángulos obtenidos de elevación y azimut, los valores máximos y mínimos de los parámetros *p0* y *p1* y el rango de movimiento que tiene cada eje de la plataforma en grados.

```
void antenna_params_calc(double azimuth, double elevation, int *p0,
int *p1)
{
      double aux=0.0;
      aux=360.0-azimuth;
      *p0 = (int) (650.0 + (aux * (1450.0/305.0)));
      *p1 = (int) (2370.0 - (elevation * (1670.0/90.0)));
}
```

Código fuente 27. Hilo de gestión del control de la antena: Función de interpolación

Por último, para enviar los parámetros al controlador de la plataforma, es necesario construir el mensaje a enviar. Dicho mensaje viene especificado en el manual de la plataforma y presenta el siguiente formato:

#0Ppos_0Sspeed_0#1Ppos_1Sspeed_1

```
…
char *message=(char*)malloc(50);
char m0[] ="#0P";
char m1[] ="#1P";
char speed[] = "S";
char aux[10];
char c_return[]="\r";
…
strcpy(message, m0);
sprintf(aux,"%d",pos_0);
strcat(message,aux);
strcat(message,speed);
sprintf(aux,"%d",speed_0);
strcat(message,aux);
strcat(message, m1);
sprintf(aux,"%d",pos_1);
strcat(message,aux);
strcat(message,speed);
sprintf(aux,"%d",speed_1);
strcat(message,aux);
strcat(message,c_return);
…
```

Código fuente 28. Hilo de gestión del control de la antena: Escritura en servos

El fragmento de código anterior forma parte de la función *build_msg()*, la cual se encuentra dentro de la librería *library_functions.c*. Dicho código es el encargado de unir todas las partes que conforman el mensaje que controla los movimientos de la plataforma.

Todo este último bloque, en el cual se realiza la transformación de las coordenadas y el envío de los comandos de control a la plataforma mecánica, se encuentra dentro de un bucle infinito. Dicho bucle se repite cada 100ms que es la frecuencia a la que se reciben las tramas desde el enlace de radio y a su vez del avión.

4.4.Thread de función de rastreo de señal

Este hilo de ejecución es el encargado de recuperar la señal proveniente del UAV en caso de que ésta se hubiese perdido. Este hilo se mantiene en un bucle infinito que se repite cada medio segundo, pero no entra en funcionamiento hasta que no se ha detectado la perdida de la señal, y siempre y cuando el ángulo inicial de referencia ya haya sido detectado.

```
while(1)
      {
            if((flag_control==lost) &&
            (flag_azimut_inicial==obtained))
            {
                  scan_signal(scanning_speed);
            }

            usleep(500000);
      }
```

Código fuente 28. Hilo de función de rastreo de señal: Bucle infinito

Una vez detectada la perdida de la señal, se llama a la función *scan_signal()*, la cual es la encargada de llevar a cabo las operaciones necesarias, con el fin de recuperar la recepción de la señal de radio. Esta función solo tiene un parámetro de entrada, *scanning_speed*, el cual se define como la velocidad de movimiento de la plataforma mecánica, durante la fase de rastreo.

La tarea básica de la función *scan_signal()*, es mover la plataforma motorizada en todo el rango del eje de azimut, de un extremo a otro de forma continua. Este proceso se lleva a cabo de forma similar a la fase de detección del ángulo inicial de referencia, esto es, se usa la varianza de los datos leídos desde la brújula digital, para detectar el momento en el que el movimiento en el eje de azimut se ha detenido, y por ende, el brazo se halla en el extremo de dicho eje.

```
while (varianza>5&& flag_control == lost){
                usleep(300000);
                compass[3]=compass[2];
                compass[2]=compass[1];
                compass[1]=compass[0];
                compass[0]=global_compass;
                varianza=varianza_muestral(compass,4);
            }
```

Código fuente 29. Hilo de función de rastreo de señal: Comprobación de varianza

Este cálculo se hace constantemente mientras no se detecte la señal, y permite que el movimiento de la plataforma pueda cambiar el sentido de giro, una vez ha llegado a uno de los extremos del eje. El siguiente fragmento de código muestra las dos condiciones *if* de movimiento a derecha o izquierda, dentro del bucle *while*.

```
void scan_signal(int speed)
{
…
      while(flag_control == lost){
            if (flag_antenna_inertia == right && flag_control ==
            lost)
            {
                  …
      while (varianza>5&& flag_control == lost){
            …
      }
                  flag_antenna_inertia = left;
            }
            if (flag_antenna_inertia == left && flag_control ==
            lost)
            {
                  …
      while (varianza>5&& flag_control == lost){
            …
      }
                  flag_antenna_inertia = right;
            }
      }
}
```

Código fuente 30. Hilo de función de rastreo de señal: Detección de sentido de giro

Una vez que la recepción de la señal se ha recuperado, el *flag_control* cambia de estado. Cuando esto sucede, se retoma la tarea principal en el hilo de ejecución *Antenna*.

4.5.Thread de función lectura de la brújula digital

Este hilo de ejecución es el encargado de gestionar la lectura de la brújula digital y empaquetarla con otras variables para su envío por protocolo UDP al equipo de monitorización. También es el encargado de llamar a la función que calcula la inercia del movimiento en el eje de azimut de la plataforma mecánica.

La lectura de la brújula se hace mediante la función *write_compass()*, que escribe un valor por el puerto serie al cual se encuentra conectada la brújula, y devuelve otro valor que corresponde con el valor del ángulo. Dicho valor se trata para darle el formato correcto y se guarda en la variable *global_compass*.

```
global_compass=write_compass(GET_ANGLE_16);
```

Código fuente 31. Hilo de función lectura de la brújula digital: Modo de lectura

El tratamiento del dato leído, se lleva a cabo en la función *read_compass()*, la cual es llamada desde *write_compass()*. Como el dato que se lee desde la brújula es de 16 bits, y la lectura del puerto serie se ejecuta en paquetes de 8 bits, el dato se lee en dos paquetes de 8 bits cada uno. Por lo tanto en tratamiento pasa por unir los dos paquetes. Al hacer esto queda un número de 4 cifras, siendo uno de ellos la parte decimal del ángulo. Por ello hay que dividir el número obtenido entre 10.

```
float read_compass(char command)
{
      …
      if (aux[0] == 0x13){

            data = read(socket_compass, b,2);
            …
            a = (b[0] <<8)|b[1];
            angle = ((float) a)/ 10;
            …
      }
      …
}
```

Código fuente 32. Hilo de función lectura de la brújula digital: Función de lectura

Una vez obtenido el valor del ángulo de la brújula digital, se empaqueta conjuntamente con los valores de localización GPS de todo el conjunto, mediante la función *Assamble_orientation()*. El objetivo de esto, como ya se ha explicado, es permitir al equipo de monitorización mostrar por pantalla la posición y orientación de la antena.

```
Assemble_orientation(origin_lat, origin_long, origin_alt,
global_compass);
```

Código fuente 33. Hilo de función lectura de la brújula digital: Ensamblar información

Por último, en este hilo de ejecución se llama a la función *antenna_intertia()*, la cual es la encargada de detectar el sentido de giro de la plataforma en el eje de azimut, esto es, derecha o izquierda. Esto se hace por un motivo lógico. Cuando la recepción de señal se pierde, el sistema entra en modo de búsqueda de señal, moviendo la base motorizada en el mismo sentido en el que se movía, justo antes de que se produjera la perdida de la señal. De este modo, existe la posibilidad de que la señal se recupere en un menor período de tiempo.

```
int antenna_inertia(void)
{
      if (times >= 5){
            if (((global_compass < old_compass-0.5) &&
            (old_compass - global_compass) <50.0) ||
               (global_compass - old_compass)>260.0){
                  flag_antenna_inertia = left;
                  old_compass=global_compass;
                  times=0;
            }
            if (((global_compass > old_compass+0.5) &&
            (global_compass - old_compass) <50.0) ||
            (old_compass - global_compass)>260.0){
                  flag_antenna_inertia = right;
                  old_compass=global_compass;
                  times=0;
            }
      }
      times++;
      return0;
}
```

Código fuente 34. Hilo de función lectura de la brújula digital: Función de inercia

Para realizar el cálculo, se comprueba el ángulo anterior con el actual, entre los cuales hay una diferencia de 5 lecturas. Dependiendo de la diferencia entre los ángulos, se modifica el valor de *flag_antenna_inertia* para indicar el sentido del movimiento. Esta función también tiene en cuenta el paso por el norte, es decir 0º o 360º. De este modo cuando el movimiento de la plataforma hace que la antena pase por ese punto, no se produce un error en *flag_antenna_inertia*, ya que en salto en el ángulo se ha tenido en cuenta en las condiciones.

CONTENIDO ANEXO B: MANUAL DE USUARIO

1. INTRODUCCIÓN

El objetivo de este manual es mostrar los pasos necesarios a un usuario que previamente posea el código fuente del programa, para que pueda compilar, ejecutar y operar con el software embebido en la antena autónoma. Por ello, se explicará donde tiene que ser ubicada la aplicación dentro de nuestra Raspberry Pi, como debe compilarse el código fuente, que cambios en la configuración estándar de la Raspberry Pi deben realizarse para el correcto funcionamiento de nuestro programa, los pasos previos que se han de seguir a la puesta en marcha de la antena autónoma, la propia puesta en marcha de la aplicación y el control mediante la interfaz gráfica del web server.

2. ALMACENAMIENTO DEL SOFTWARE Y COMPILACIÓN

En primer lugar, es necesario ubicar la carpeta "UAVAntena" que contiene el software de nuestra aplicación en la raíz de la Raspberry Pi para su posterior ejecución automática.

Para ello debemos tener en cuenta una serie de pasos con el objetivo de tener los derechos de administrador para realizar estas modificaciones:

1) Abrir el "Root Terminal" (FOTO de menú de inicio) o bien abrir el terminal estándar y escribir todas las sentencias precedidas de la palabra "sudo".
2) Ejecutar el "Gestor de Archivos" desde el terminal escribiendo:

```
>> sudo pcmanfm &
```

3) Navegar con el Gestor de Archivos y copiar la carpeta "UAV Antena" en el directorio raiz "/".
4) Para habilitar todos los derechos de lectura, escritura y ejecución de los archivos de nuestro programa, debemos introducir la siguiente línea de comandos por terminal:

```
>> sudo chmod –R 777 /UAVAntena
```

Una vez ubicado el programa en nuestra carpeta, se procede a compilar los archivos tanto del código en C como del servidor web en Python.

- Para compilar el código del programa en lenguaje C basta con escribir en el terminal la siguiente línea dentro del directorio "/UAVAntena":

```
>> gcc –lpthread –o main main.c -lm
```

- Para generar el archivo ejecutable para el servidor web en Python, la Raspberry Pi viene por defecto con un intérprete instalado de Python. Bastará con escribir nuevamente en el terminal dentro de nuestro directorio la siguiente línea y automáticamente se generar dentro de la carpeta todos los archivos compilados necesarios para el servidor web:

```
>> python mainweb.py
```

3. CONFIGURACIÓN PREVIA DE LA RASPBERRY PI

En el sistema operativo Linux instalado de serie en la Raspberry Pi deben realizarse una serie de modificaciones específicas en los registros para el correcto funcionamiento de nuestra aplicación.

- Añadir la ruta del ejecutable del servidor web en el archivo de arranque de la Raspberry Pi con el objetivo de poner en marcha automáticamente el servidor web al encender el sistema. Hay que copiar al final del archivo justo antes de *exit 0* del directorio "/etc/rc.local" las siguientes líneas:

```
cd /UAVAntena
sudo Python mainweb.py &
exit 0
```

  Para editar este archivo basta con ejecutar en la línea del terminal la sentencia "sudo nano /etc/rc.local" y salir guardando los cambios realizados con el comando "CTRL-X".

- Para la correcta comunicación de la Raspberry Pi con la brújula digital mediante la salida de pines destinados a la comunicación serie del GPIO, es necesario modificar del mismo modo que se modificó el archivo anterior, el registro "/boot/cmdline.txt" de modo que quede como se muestra en el siguiente cuadro de texto:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 rootwait
```

- Modificar el modo de asignarle una dirección IP fija a la Raspberry Pi establecida como IP dinámica por DHCP por defecto. De este modo nuestro servidor web siempre estará accesible en la misma dirección de nuestra red local, en este caso fijada como la 192.168.2.150. Hay que modificar el registro "/etc/network/interfaces" con el editor "nano" igual que en los dos casos anteriores de modo que quede de la siguiente forma:

```
auto lo
iface lo inet loopback
iface eth0 inet static

address 192.168.2.150
gateway 192.168.2.1
netmask 255.255.255.0
```

Una vez se han realizado las modificaciones explicadas, es necesario reiniciar la Raspberry Pi para que los cambios surtan efecto.

4. PUESTA EN MARCHA DE LOS SUBSISTEMAS

Antes ejecutar la aplicación hay que dejar correctamente conectados todos los subsistemas hardware.

- Módulo de radio con comunicación serie con la Raspberry Pi, conectado con cable RS-232/USB a una de dos las salidas del hub USB de la Raspberry Pi.
- Controlador de los servos con comunicación serie también con la Raspberry Pi, conectado con cable RS-232/USB a la otra de dos las salidas del hub USB de la Raspberry Pi.
- Brújula digital con comunicación serie con la Raspberry Pi, conectada con pines de la UART (Tx, Rx), tierra y alimentación a los pines del GPIO específicos de la Raspberry Pi.
- Router wifi con cable Ethernet RJ-45 conectado a la clavija Ethernet de la Raspberry PI.

La última conexión necesaria es la de la alimentación de 5 V con un adaptador común de entrada micro-USB. Para poner en marcha la Raspberry Pi únicamente debemos conectarla a la red y automáticamente arrancaría. En la siguiente figura se muestran todas las conexiones necesarias para nuestra aplicación.

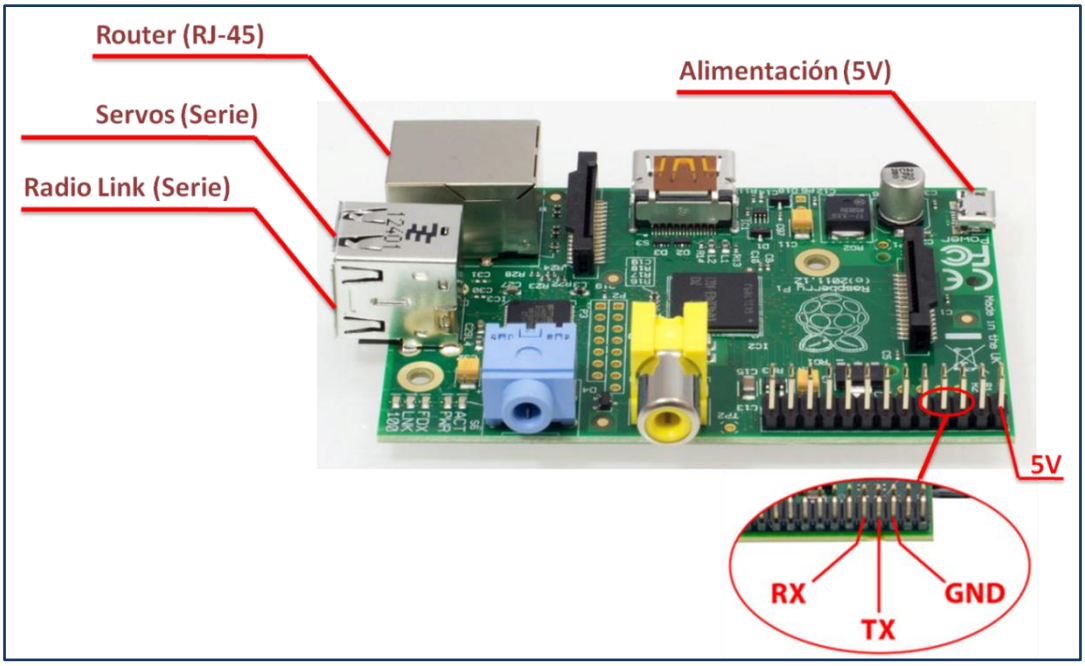

Figura 40. Conexiones a realizar en la Raspberry Pi

5. EJECUTAR LA APLICACIÓN

Una vez todas las conexiones están correctamente realizadas, la Raspberry Pi está correctamente configurada como se explica en el apartado 3 y el software embebido está almacenado en memoria correctamente, se puede proceder al arranque del sistema y ejecución del programa. Para ello seguiremos los siguientes pasos:

1) Conectar tanto la alimentación del Router wifi como la Raspberry Pi a la red.
2) Activar el interruptor verde de alimentación de la base robotizada de la antena de alta ganancia.
3) Transcurridos unos segundos, el sistema habrá tenido tiempo de cargar toda la configuración correctamente y será posible visualizar del servidor web desde cualquier ordenador conectado previamente a nuestra red local wifi abierta "Edimax". Accediendo a la dirección "192.168.2.150" desde un navegador web se mostrará la interfaz gráfica con nuestro programa desde donde será posible ejecutarlo, pararlo y apagar el sistema.

6. CONTROL DESDE LA INFERFAZ GRÁFICA

Una vez hemos abierta la interfaz gráfica ofrecida por nuestro servidor web, se explican cuales son las distintas posibilidades existentes para que el usuario pueda controlar la antena.

Antes de ejecutar el programa con el botón "TELEMETRY" los campos de los parámetros de entrada deben estar rellenos. En caso, contrario el programa no arrancara por falta de argumentos de entrada.

- Introducir localización de la antena (latitud, longitud y altura).

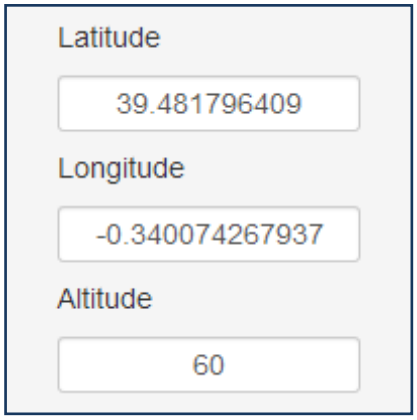

Figura 41. Web server: Parámetros localización antena

- Introducir time-out de inicio y velocidad de giro de operación de rastreo.

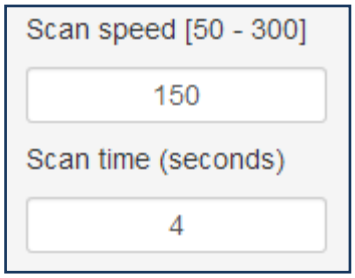

Figura 42. Web server: Parámetros función rastreo

- Introducir direcciones asignadas dentro de nuestra red local de los dispositivos de monitorización a conectar. En caso de que no se introdujese ninguna dirección válida (192.168.2.X con X ϵ [100,254]), el envío de paquetes se realizaría automáticamente por *broadcast* (192.168.2.255).

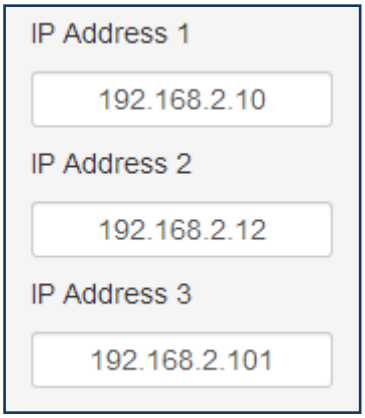

Figura 43. Web server: Direcciones IP

- Actualizar los datos introducidos en los campos pulsando el botón “Update” y poner en funcionamiento el programa con el botón “Telemetry”.

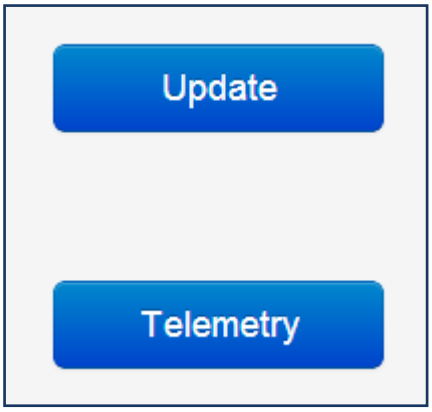

Figura 44. Web server: Actualizar datos y arrancar

- Existen otros tres botones para poder parar la aplicación con “Stop”, reiniciar el sistema completo con “Reboot” o apagar el sistema con “Shut Down” (Es importante no desconectar la alimentación de la Raspberry Pi al menos 30 segundos después de apagar el sistema).

Figura 45. Web server: Reiniciar, Parar o Apagar